高等学校试用教材

FORTRAN 语言程序设计

谭浩强　田淑清　编著

高等教育出版社

内容提要

本书主要讲授 FORTRAN77 的基本内容和程序设计方法，该书选材适当，叙述全面细致，文字通俗易懂. 全书贯穿结构化程序设计思想，有很丰富的例题，便于自学.

本书可作为高等院校和电视大学计算机专业和非计算机专业的教材，也可供使用计算机的科技人员及管理人员自学参考.

图书在版编目(**CIP**)数据

FORTRAN 语言程序设计/谭浩强　田淑清编著. —北京：高等教育出版社，1987.1(2001 重印)

ISBN 7-04-000758-4

Ⅰ. F…　Ⅱ. 谭…　Ⅲ. FORTRAN 语言-程序设计-高等学校-教材　Ⅳ. TP312

中国版本图书馆 CIP 数据核字(97)第 27982 号

出版发行	高等教育出版社	购书热线	010-64054588
社　　址	北京市东城区沙滩后街 55 号	免费咨询	800-810-0598
邮政编码	100009	网　　址	http://www.hep.edu.cn
传　　真	010-64014048		http://www.hep.com.cn
经　　销	新华书店北京发行所		
印　　刷			
开　　本	787×1092　1/16	版　　次	1986 年 10 月第 1 版
印　　张	16.75	印　　次	年　月第　次印刷
字　　数	382 000	定　　价	13.70 元

前　言

电子计算机的出现是当代科学技术发展中一件具有划时代意义的重大事件.它标志着人类正在向信息化的社会前进.

为了赶超世界先进水平,必须在我国大力推广普及计算机的应用.为了掌握计算机,必须系统地学习计算机的知识.现在,计算机教育已成为高等学校中的基础教育之一.实践证明,在高等学校中以计算机高级语言作为学习计算机知识的入门,是一条捷径.

FORTRAN 语言是国际上最早出现的计算机高级语言,得到了广泛的应用.它适用于数值计算.在科学技术发达的国家中,几乎每一个工程技术人员都会使用 FORTRAN 语言.在我国大多数理工科院校中也都开设了 FORTRAN 语言课,本书就是适应这种要求而编写的.

本书采用 FORTRAN 语言的最新标准——FORTRAN77.国内大多数计算机都已配置了 FORTRAN77编译系统,国外的许多科技书籍中,都有用 FORTRAN77 编写的程序.因此,我们应当学习和掌握 FORTRAN77.考虑到国内少数单位的计算机系统上还没配置 FORTRAN77,以及社会上还在继续使用着大量的 FORTRAN Ⅳ程序,本书中对 FORTRAN Ⅳ的语句也作简要的介绍.本书中的例题是用 FORTRAN77 语句写的,同时说明了把它改写为 FORTRAN Ⅳ程序的方法,以便在那些还未装 FORTRAN77 编译系统的计算机上运行程序时不致发生困难.

结构化程序设计方法是近年来兴起的一种程序设计的新的方法.它使程序有良好的结构,使之易于编写、易于阅读、易于修改和调试,能大大提高程序设计的效率和质量.本书介绍了结构化程序设计的方法和结构化流程图,并在全书例题中采用了结构化的方法编写程序,以便使读者掌握新的程序设计思想和方法.

1985 年我们曾编写了一本《FORTRAN77 结构化程序设计》(高等教育出版社出版),供高等院校高年级学生和研究生使用.在该书中介绍了数值运算和非数值运算的算法,结合例题介绍了一些数据结构的知识.该书出版后受到读者的欢迎.根据许多高等院校和读者的建议,我们在该书的基础上重新编写了本书,供高等院校低年级和非计算机专业学生使用.在本书中不涉及到较深的高等数学内容和复杂的数据结构,大学一年级下学期的学生就可以学懂本书的基本内容.

在本书中只介绍最基本的数值运算和非数值运算的算法,以及程序设计的基本知识.有了这个基础,以后继续提高再去解决复杂的运算是不会太困难的.语言是一个工具,应该在使用过程中熟悉它,掌握它.希望读者不要把它作为一种“理论知识”来学,以“知道了”为满足,而应当以会应用为目的,做到灵活地、得心应手地用语言编写程序并解决实际问题.

几年来,我们编写了一些计算机语言的书籍,对教学体系和教材的编写方法进行了一些改革.主要的目的是使广大初学计算机的同志能尽快地掌握计算机的基本知识,进入计算机应用领

域.我们根据多年教学实践的经验,研究总结了初学者的特点,根据他们的认识规律,循序渐进地提出问题和解决问题,步步深入.不从枯燥的规则定义出发,而从具体问题入手经过分析引出结论.通过大量的例子分析算法,介绍程序设计的基本方法和技巧.既注意了教材的系统性、科学性,又注意了易读性、启发性.努力把被认为枯燥无味的"语言"讲得生动活泼、使人感兴趣.实践证明,这是行之有效的方法.本书就是按照以上原则编写的.重点不放在语法规则的叙述上,而放在算法和程序设计方法上.从一开始就介绍程序,要求读者编写程序.以后每学新的一章都要运用所学的内容去编写程序,通过反复编写和运行程序来掌握语言的规定和程序设计的方法.希望读者在学习本书时,不要把主要精力放在死记语法规则上(靠死记是记不住的),而要学会怎样把各个孤立的语句组织成一个有机的、好的程序.

学习本课程除了听课和自学外,一定要强调多做练习,多编程序,多上机调试程序,否则是学不好的.

FORTRAN77 标准分为全集和子集,子集只包括全集中的一部分(基本的)功能.一般微型计算机系统配置的是 FORTRAN77 的子集.本书既介绍 FORTRAN77 子集又介绍 FORTRAN77 全集的主要功能,便于读者在不同的计算机系统上都能使用.

FORTRAN77 便于编写结构化的程序,但仍保留了 FORTRAN66 标准中的一些非结构化语句,这只是为了与 FORTRAN66 兼容,以便能使用过去编写的 FORTRAN 程序.在第十一章中介绍了这些语句和其他一些不常用的语句,读者需要时可以查阅,但建议编程序时尽量不用它们.第十二章介绍了"文件",在使用大量数据时,数据文件是很有用的.由于学时的限制和初学者的基础,本书只介绍文件的基本知识和简单的应用例子.我们认为,在读者对 FORTRAN77 的基本语句的使用比较熟练编写程序有一定经验后,在遇到需要用文件处理比较复杂的问题时,这时再进一步深入学习有关文件的内容,效果更好一些.

为了便于一些缺乏师资的单位开展教学,已由清华大学电教中心将本书的内容录制成录像带,由谭浩强主讲,共 36 学时.

本书由谭浩强、田淑清编写.其中谭浩强编写第一、二、三、四、五、六、八章,田淑清编写七、九、十、十一、十二、十三章.在出版前,清华大学计算机系林行良副教授曾审阅了本书,提出了一些很好的意见,不少兄弟院校的教师对本书的出版给予了支持和帮助,特此一并表示感谢.

本书一定会有不少缺点或错误之处,恳请同行和读者多提意见.

编者

1986.4.1

目　录

第一章　计算机、算法和程序设计

人类正在向信息化的社会前进，电子计算机的出现影响着人类的生产方式和生活方式，可以说，计算机打开了通向信息化的大门. 当今，计算机迅速地进入几乎一切领域，成为人们处理种种复杂任务所不可缺少的现代工具. 可以毫不夸张地说，没有计算机就没有现代化.

在未来的社会里，每一个有知识的人都应当会使用计算机. 计算机知识应当成为新一代知识分子知识结构和智能结构的一个重要组成部分. 为了使用好计算机，必须首先了解计算机.

§1.1　信息处理和电子计算机

人类在自己的活动中最早认识的是“物质”，后来又发现了“能量”，到了二十世纪才认识到“信息”是维持人类社会活动、经济活动和生产活动的资源之一，是构成客观世界的不可缺少的要素之一.

其实，人类自古就开始与信息打交道，进行信息处理. 例如，用棍石计数和用算盘计算就是对数据进行记载和运算的例子，财务记账也是一种信息处理. 什么是信息？简单地说，信息是表现事物特征的一种普遍形式，这种形式应当是能够被人类和动物的感觉器官（或仪器）所接受的. 例如以声音、图像、文字、指纹等形式表示出来的内容都是信息. 人们看一部电影或一本书，就获得了信息. 确切地说，信息是客观存在的一切事物通过物质载体所发生的消息、情报、指令、数据、信号中所包含的一切可传递和交换的知识内容. 不同的事物有不同的特征，不同的特征会通过一定的物质形式，如声波、文字、电磁波、颜色、符号、图像等发出不同的信息（消息、情报、指令、数据、信号等）. 信息是与物质运动紧密相联系的. 自然界、人类社会、人类思维活动中自始至终存在着信息运动. 人类的知识就是一种特定的信息，是人们对客观世界的信息经过大脑的思维加工而形成的系统化的信息.

人们对信息处理的方式取决于所处时代的生产发展和科学技术水平，但是又反过来对社会的发展产生巨大的影响. 在科学技术迅猛发展的今天，信息量如此之大，很难想象用人工的方法能够最大限度地、准确地、迅速地处理人们所需的信息以满足需要. 据统计，人类的科学知识，在19世纪每五十年增长一倍，20世纪中叶每十年增长一倍，20世纪70年代每五年增长一倍，各种新的学科层出不穷，面临这种知识激增的时代，必须采用现代化的处理信息的工具. 于是电子计算机就应运而生了.

所谓信息处理指对信息的收集、加工、存储、检索等，其中“加工”包括计算、排序、归并、制表、摸拟、预测等操作. 信息处理可以分为两大类，即：数值处理（又称数值计算）和非数值处理（或称非数值运算）. 计算机的应用大体可以归纳为五个方面，即：数值计算（科学计算），事务管理（例如企业、银行、计划管理等），工业控制，计算机辅助设计，以及人工智能（用计算机模拟人

脑的部分活动）. 当然还可以有其他的用途.

有人以为“计算机”的作用就是“计算”，这是一种误解. 的确，早期的计算机主要用于数值计算，用来解决工程技术问题中一些复杂的计算. 但是由于社会发展的需要以及计算机科学技术的发展，计算机在非数值处理方面的应用得到了迅速的发展. 目前，计算机在非数值处理方面的应用已经大大地超过了它在数值处理方面的应用. 从这个意义上说，“计算机”这一名词已经不能确切地反映出它的本质和作用. 其实计算机是一种现代化的处理信息的工具，称之为“信息处理机”更为确切，也有人称之为“电脑”. 只是由于习惯上的原因，我们仍称为“计算机”，但是应当对计算机的作用有一个全面的理解.

电子计算机最早产生于 1946 年，至今不过四十年的历史，但是它的发展十分迅速. 它经历了四个发展阶段：电子管计算机（第一代）；半导体计算机（第二代）；集成电路计算机（第三代）；超大规模集成电路计算机（第四代）. 与之同时，计算机的软件也有了迅速的发展.

这四代计算机基于同一个基本原理，就是以二进制和程序存储控制为基础的结构思想. 这个思想是由美籍数学家冯·诺依曼（**Von Noumann**）于 1946 年最早提出来的，它确立了至今为止的各代计算机的基本工作原理. 根据这个原理，信息在计算机内部以二进制数表示，除了要将运算所需的数据输入计算机以外，还要将运算的步骤事先编成指令（指令也是用二进制数表示的），将指令输入到计算机内储存起来，这就是“存储程序”的概念. 计算机根据人们事先存储在计算机里面的程序指令，一步一步地进行操作，对数据进行加工处理以及输入输出. 从这个意义上说，计算机对信息的处理是不必人们干预的，也就是说是“自动”的，由程序控制的. 因此，现在的计算机归根到底还是根据人们预定的意图工作的. 这种基于“存储程序”原理的计算机，被称为冯·诺依曼型计算机.

这种计算机的特点归纳起来有以下几点：

1. 电子的. 计算机的工作基于电子脉冲电路原理，由电子线路产生电脉冲，依靠它进行数据传送和运算. 电子计算机的运算速度取决于电子线路. 从理论上说，计算机运算速度只受到电的传播速度的限制. 所谓计算机运算速度，是指一秒钟能够存取指令的数目. 假如有一台计算机，它具有一秒钟存（或取）一万条指令的能力，这台计算机的运算速度就是一万次/秒. 一般微型计算机的运算速度为每秒几万次到十几万次. 我国研制成功的“银河”巨型计算机的运算速度为每秒一亿次. 国外已有每秒几亿次的计算机.

2. 具有内部存储能力. 为了存放数据和指令，计算机中有存储器（磁蕊存储器或半导体存储器），可以存入若干电讯号，代表数据或指令. 从计算机内部的存储器存取数据和指令，就可以大大降低数据处理的时间，并使程序控制成为可能.

3. 由程序自动控制计算机的操作. 在计算机运行期间，由预先编好并存入计算机的指令指挥计算机的工作.

因此，电子计算机是一种以高速进行操作、具有内部存储能力、由程序控制操作过程的自动电子装置.

正在研制的第五代计算机将是一种非诺依曼型的计算机，它采取完全新的工作原理和体系

结构.它更接近于人们思考问题的方式,即“推理”方式.第五代计算机不仅在其采用的技术与以前不同,而且在概念和功能方面也不同于前四代计算机.这种新型的计算机被称为“知识信息处理系统”,其功能从目前单纯的数据处理发展到知识的智能处理.这些新型计算机具有人工智能的功能.因此,未来的第五代计算机的研制成功将是对计算机科学技术的一项突破性的贡献,被称为“第二次计算机革命”.目前许多国家都投入了大量的人力财力研制第五代计算机.但是从目前情况看,第五代计算机研制成功并真正投入使用,并非很短时间内所能实现的.作为计算机的应用者,目前我们还是应该学习、熟悉和使用好第三、第四代的计算机.

§1.2 电子计算机的组成

按照冯·诺依曼的理论,电子计算机应当具有输入输出、计算、存储、判断以及内部控制的功能,这些功能分别由输入设备、输出设备、运算器、存储器以及控制器等几个基本部件组成的.图1.1是一个中型计算机系统,图1.2是微型计算机系统.

一、存储器

存储器是计算机的一个重要设备,用来存放数据和指令.过去的计算机主要用磁芯存储器,近年来已多改用半导体存储器.由于用的电子器件有两个稳定工作状态(截止和接通,有脉冲和无脉冲),可以用它们分别代表二进制中的0和1.每一个能代表0和1的电路称为一个二进制位(bit)或称为比特.一个存储器就是由千千万万个这样的二进制位电路组成的,它好比一个大

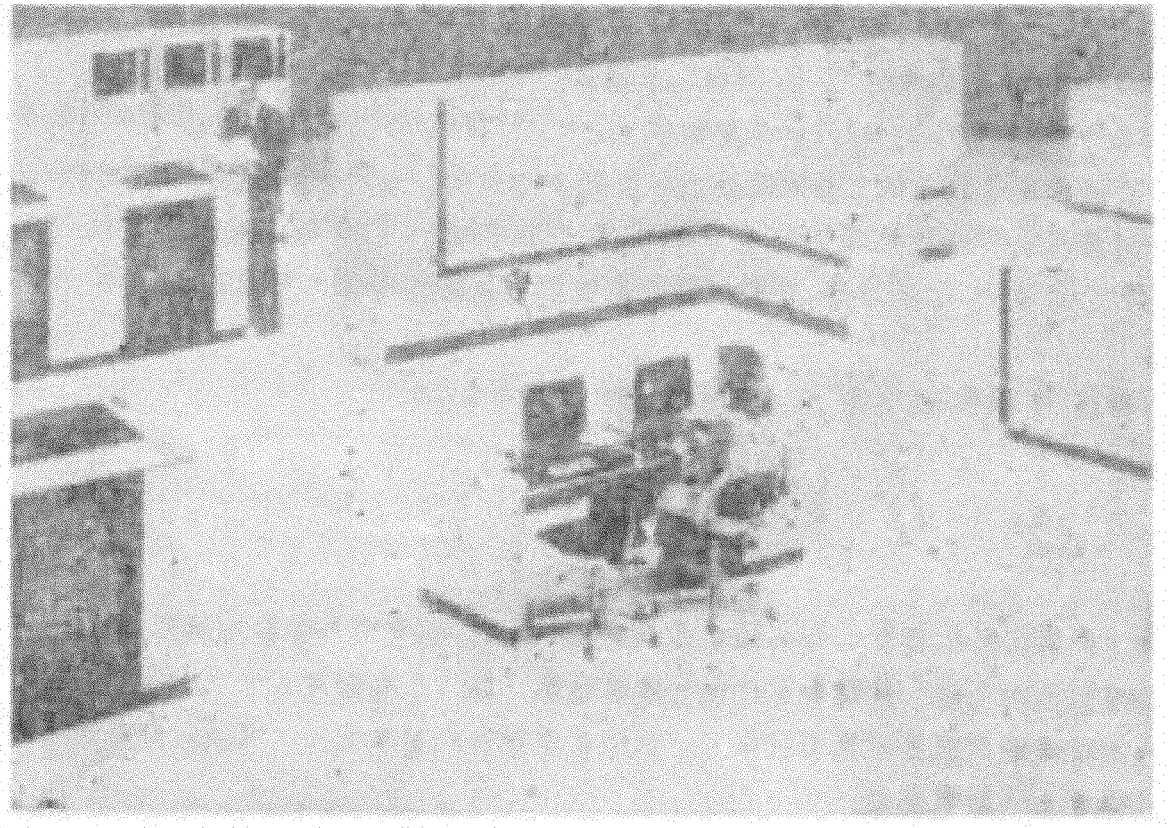


图1.1

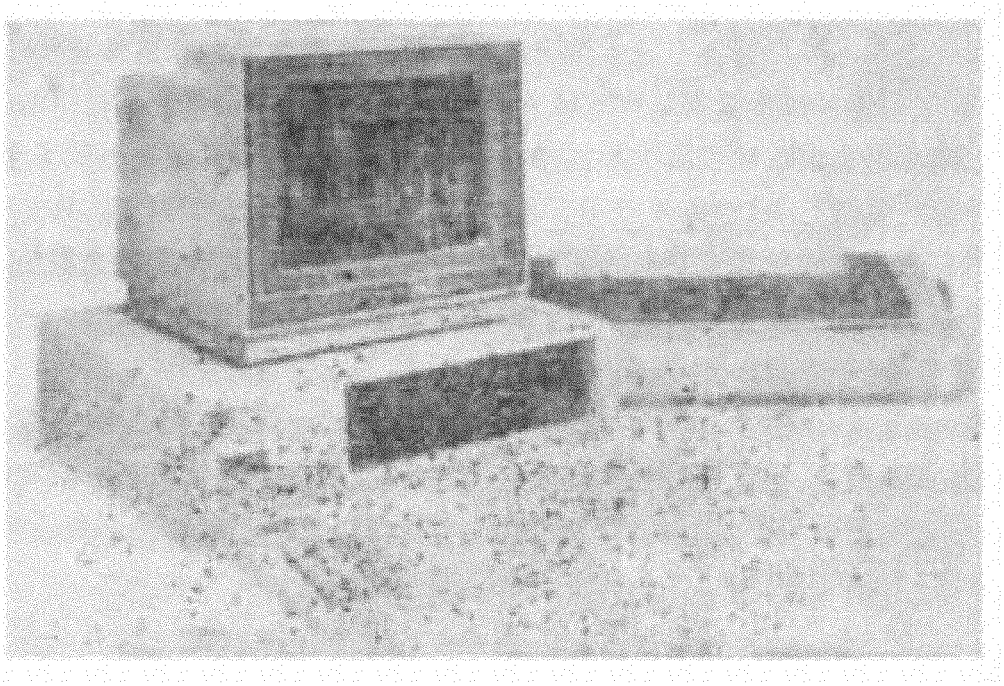


图 1.2

仓库，可以容纳亿万个二进制位信息.

为了管理上方便，把一个存储器分为若干个单元，就如同一个大饭店分为许多小房间一样，在每一个“小房间”中存放一条指令或一个数据，这些小房间称为“存储单元”. 一个存储单元包含若干个二进位. 每一种计算机可以分别规定一个存储单元包含多少个二进位(bit). 有的微型计算机的一个存储单元包含 8 个二进位，有的则包含 16 位，还有的是 32 位. 通常把存储单元称为“字”(word). 常说的“字长 16 位”，指的是一个存储单元包含 16 个二进位. 一般把八个二进位称为一个“字节”(byte). 一个存储单元(一个计算机“字”)由一个或几个字节组成. 也就是说，一个存储单元所含的二进位的数目一般是 8 的倍数. 存储器的容量以字节为单位计算. 一般微型计算机的存储容量为 16K 至 64K 字节(1K 代表一千，实际上应该是 $2^{10}=1024$). 有的微型计算机的容量为 512K 字节，高档微型计算机的容量甚至可达一兆到几兆(1 兆 $=10^6=1000$K)字节.

存储器、存储单元、字节、位之间的关系可以表示如下：

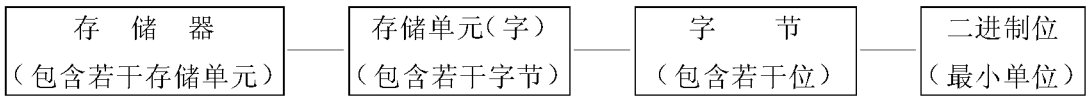


图 1.3

为了向指定的存储单元存入信息或从中取出信息，需要对存储单元编号，如同旅馆中的房间必须有房间号码一样. 存储单元的编号称为地址. 每一个存储单元有一个地址，在存入信息时，根据指出的地址找到所需的存储单元，把信息存到该存储单元中. 也可以根据指定的地址从特定的存储单元中取出信息.

请读者不要将存储单元的地址和存储单元的内容这两者混淆. 前者是存储单元的标志，用来

查找指定的存储单元，后者就是所需存取的信息. 见图 1.4

图 1.4

计算机存储器有一个特点：在信息被“取”出之后，存储单元的内容不会改变，只有在向同一个存储单元存入一个新的信息时，原有的内容才会被代替. 因此，“存”、“取”实际上应当理解为“写入”和“读出”.

二、运算器

运算器是对信息进行加工的重要部件. 计算机的各种运算都是由运算器完成的. 应当强调指出. “运算”不仅指算术计算，而且还包括逻辑运算（泛指非算术性的运算. 例如，比较两个数值，根据比较结果决定操作的内容）. 如果需要将两个数值进行算术运算，应先从存储器的两个存储单元中，将两个数据取到运算器中，运算完毕后结果仍在运算器中，可以根据需要将它送到某一存储单元中保存起来，以备后用.

三、控制器

计算机的各种操作都是在控制器的指挥下进行的. 控制器根据事先存入计算机的指令向运算器、存储器、输入输出设备发出控制信号，使之按要求进行工作. 它是计算机的“神经中枢”.

运算器和控制器合称为中央处理器（Central Processing Unit），简称 CPU，通常把它们制造在一个晶片上，它是计算机的核心部分，存储器好比仓库，自己是不能进行和完成任何操作的，操作是由 CPU 控制并完成的.

四、输入输出设备

所有需要由计算机处理的信息都是通过计算机的输入设备送到计算机的存储器中的. 在需要对该信息进行运算时再从存储器取到运算器中　常用的输入设备有：卡片读入机（读卡机）、纸带输入机、终端显示器等. 输入设备能自动地将输入的信息转换成二进制形式存到存储器中. 例如，在终端显示器的键盘上按下字母键“A”，按 ASCII 代码的规定（一般计算机系统采用 ASCII 代码，见附录 IV），“A”的代码是二进制码 01000001，显示器设备将 01000001 这个信息输入计算机.

计算机内的信息可以通过输出设备传送出来，例如，可以将计算结果打印在纸上，显示在荧光屏上，或存储在磁盘上. 常用的输出设备有：打印机、终端显示器（它既是输入设备，也是输出设备，用键盘输入，从荧光屏上输出）、绘图机等. 在输出时，输出设备会自动地将计算机内的二进制信息转换成人们所需的字母、数字或图形.

大批量的信息常记录在磁带或磁盘上，在需要的时候通过磁带机或磁盘机把信息输入到计算机中去. 磁带机常用作大中型计算机的输入输出设备. 在大、中型计算机系统中常使用硬磁盘，一个磁盘组（包含 10 个左右硬盘）的容量可达几百兆字节. 微型计算机常用软磁盘，软盘有 5 英寸和 8 英寸两种，容量为几百 K 字节. 在一些高档微型机系统中也使用一种称作温盘（Winches-

ter 盘)的硬磁盘,容量约为 10 兆字节. 磁带和磁盘可以记录大量的信息,因此又作为计算机的外存储器使用. 例如,一个单位的人事、工资档案可以记录在一个磁盘上,便于保存和管理,需用时临时找出来输入计算机,即可实现检索或其他处理.

计算机系统示意图和各部分之间的关系见图 1.5 和图 1.6.

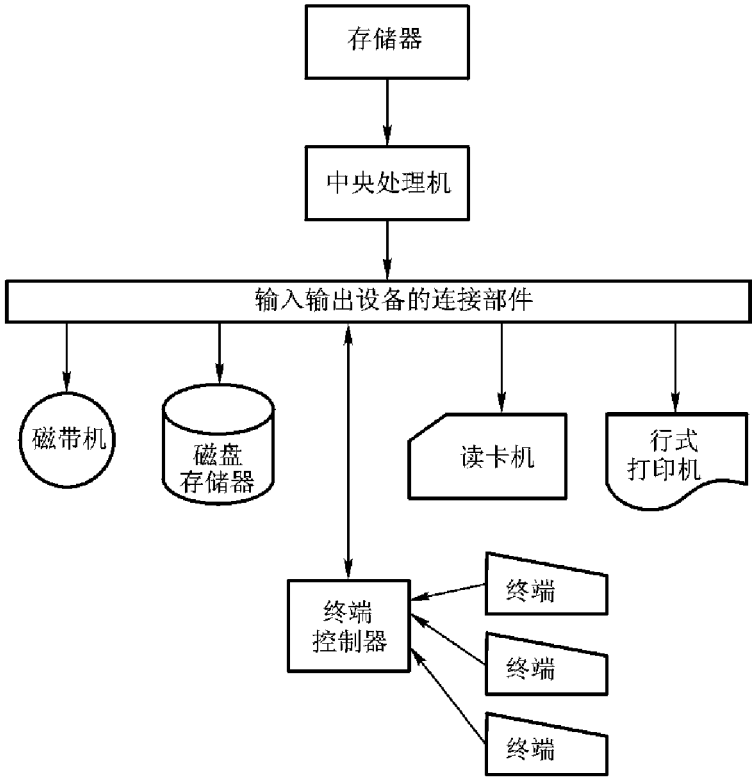


图 1.5

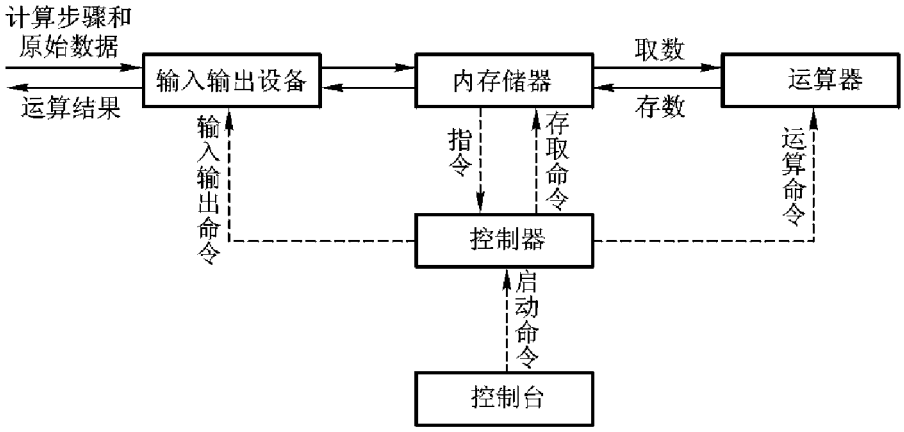


图 1.6

§1.3 计算机语言和软件系统

人们要使计算机按自己的意图工作,必须使用计算机所能接受、理解和执行的指令. 从根本上说,计算机只能接受和处理由二进制代码组成的指令. 每一种型号的计算机在设计时由设计者规定好一套指令系统,确定用什么指令来完成什么操作. 例如,在某种计算机上用

0000101011011101 来实现一次加法. 这种计算机能接受的代码称为机器指令，它是面向机器的. 机器语言是机器指令的集合. 要让计算机产生一系列的操作，应当写出一条一条的机器指令. 为解决某一问题而设计的一段指令序列称为程序.

用机器语言编程序是一件繁琐而枯燥的工作，而且直观性差，不同计算机的指令系统互不通用. 这给计算机的广泛使用造成很大的不便.

符号语言　比机器语言前进了一步，它用符号代替二进制码. 例如用 MOV 代表“传送”，ADD 代表“加法”，一条符号语言指令对应于一条机器指令. 当然，计算机不能直接接受和执行符号语言程序，必须把它一条一条地“翻译”成机器指令，这个翻译工作叫做代真. 最初的代真是由人工完成的，后来人们编制了一个程序让计算机来完成代真. 这个起代真作用的程序称为符号汇编程序或汇编程序.

符号语言和机器语言一样，都是依赖于具体机器的. 因此它们被称为低级语言. 对初学者来说，用符号语言编写程序仍然不是一件容易的事情.

后来人们创造了“高级语言”，又称“程序设计语言”. 它不是面向机器的，而是面向问题的. 即不依赖于具体机器. 用高级语言写的程序可以适用于任一种类型的计算机（或者只需作很小的修改）. 高级语言与人们习惯使用的自然语言和数学语言比较接近，因而人们容易理解和使用. 例如，用 READ 表示“读入”数据，用 PRINT 表示“打印”，用 A =（B + C）/2 表示计算$\frac{B+C}{2}$的值并将结果存到变量 A 中. 显然，用高级语言写程序比用低级语言写程序容易得多，任何一个具有中学以上文化程度的人都能很快学会和使用它.

目前已经出现了几百种高级语言，各有不同的语法规定和用途，常用的高级语言在下页表中列出.

和符号语言不同，高级语言的一个指令（通常称为语句）不是对应于一条机器指令，而对应若干条机器指令. 例如可以在一个语句中包括加、减、乘、除. 转移等多种操作. 显然，高级语言程序比符号语言程序简单易写.

同样，计算机不能直接接受高级语言程序，而必须首先翻译成机器指令. 这个工作比汇编代真复杂得多. 将高级语言程序（称为源程序）转换为机器语言程序（称为目的程序或目标程序）的工作由“编译程序”来完成. 其工作过程见图 1.7.

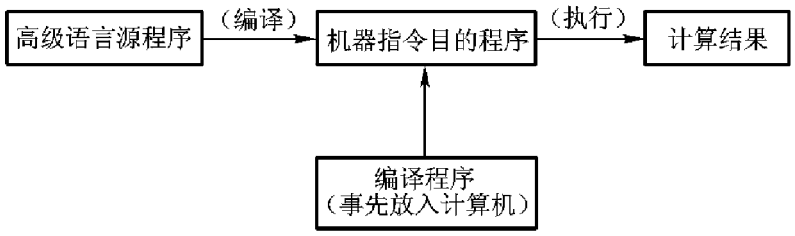


图 1.7

应当指出，并不是任何机器在任何时候都能运行任何一种语言程序的. 如果想在一台计算机

上运行一个 FORTRAN 语言程序，必须首先保证该计算机具备 FORTRAN 编译程序. 一般在计

名　称	缩写字含义	发表日期	主要应用	评　　语
FORTRAN	FORmula TRANslation	1956	科学计算	第一个被广泛采用并且目前仍在广泛使用的语言
COBOL	COmmon Business-Oriented Language	1960	商业，数据处理	风格似英语，是应用最广泛的一种语言
ALGOL 60	ALGOrith mic Language 1960	1960	科学计算	适于数值计算，包括逻辑处理，它的前身 ALGOL 58 是几种重要语言的基础
LISP	LISt Processing	1960	表处理	用于人工智能研究
BASIC	Beginners All Purpose Symbolic Instruction Code	1964	数值计算	简单易学，是大多数微型机和个人计算机都配备的语言，目前应用范围已超过数值计算，在管理领域也得到广泛使用
PL/1	Programming Language/1 但也有人认为 PL/1 不是缩写字	1964	多用途	第一个极大型的、功能极强的语言，包括了 ALGOL，COBOL，FORT-RAN 和其他一些语言的概念和特点
ALGOL 68	ALGOrithmic Language 1968	1968	多用途	功能非常强的语言，但不能和 ALGOL 向上兼容
PASCAL	pascal 是十七世纪法国哲学家 Blaisl pascal 的名字，他发明了计算器	1971	多用途	小而精练，具有很多特色的语言. 广泛用于程序设计教学，是结构化的语言
C	非缩写字	1975	系统程序设计	用于编写 UNIX 操作系统和它的大多数应用软件
APA	非缩写字，Ada 是一人名	1979	多用途	是功能很强的语言，被称为 20 世纪 80 年代的语言

算机出厂和出售时，提供有关的编译程序和其他软件（存放在磁盘或磁带上提供给用户）. 在运行某一语言的程序前，把编译程序输入计算机内，并用它对高级语言源程序进行翻译，然后才能使计算机执行目的程序.

可见，计算机必须配备各种有专门用途的程序（如汇编程序、编译程序等）才能有效地工作. 一个好的计算机系统除了要有质量高的设备（包括计算机主机和输入输出设备. 它们是机器系统，称为硬件）外，还应当有功能完善的程序系统（称为软件）. 软件的作用是更好地发挥机器系统的作用. 一个不配备任何软件的计算机（称为“裸机”）的作用是有限的，使用是极不方便的，效率是很差的. 近年来，各计算机厂研制软件所花费的人力和财力远远超过硬件. 在购买计算机时也必须选购各种必须的软件.

除了上面介绍的编译程序、汇编程序外，软件还包括操作系统、诊断程序、数据库管理程序、程序库等.

操作系统（Operating System，简写为 OS）是一个十分重要的软件，是整个软件系统的核心. 它是一个庞大的管理程序，其作用是控制所有在该计算机上运行的程序并管理这个计算机的所有

资源.它能充分利用计算机的全部资源(包括硬件和软件资源),最大限度地发挥计算机系统各部分的作用.例如,一台中型计算机允许几十个用户同时使用计算机(分时方式),操作系统的作用有如"总调度",使各程序轮流使用计算机的 CPU,使计算机的各部分(包括输入输出设备)协调有效地工作.一台中型计算机有了良好的操作系统后,利用率可提高数十倍.

图 1.8 表示计算机——操作系统——编译程序——用户源程序间的关系。

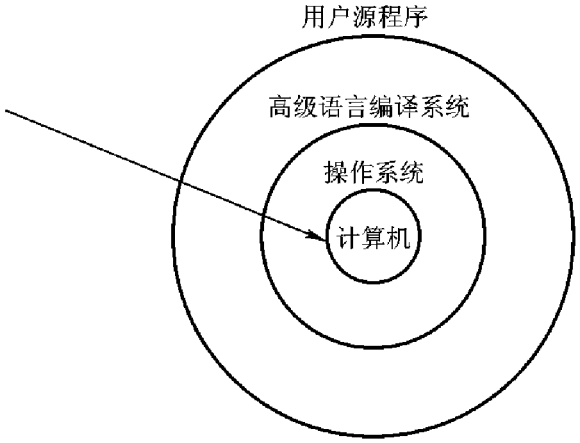


图 1.8

§1.4 计算机算法

为了解决一个问题而采取有限的步骤,称为算法.例如建造房屋的各工序可以称为建造房屋的算法.广义地说,任何一个问题的进行过程都有它自己的算法.当然我们只讨论计算机算法,即如何使计算机一步步地进行工作的具体过程.我们用机器语言或高级语言编写成程序,指挥计算机按指令工作,这就是一种计算机"算法".在设计一个计算机算法时,应当考虑到计算机能否执行.例如让计算机完成"打印 8 +5 的和"是可能的,而让它"替你配一副眼镜"是不能实现的(至少目前如此).

下面举一例说明如何设计一个计算机算法.

例 1 商店结账,要求将当天 100 笔收入累加,打印出总和,应当使算法的每一步在计算机上都是能够执行的.可以写出下面的算法:

(1) 将第一笔收入输入给计算机;

(2) 将第二笔收入输入给计算机;

(3) 将以上两笔收入相加;

(4) 将第三笔收入输入给计算机;

(5) 将它和前二笔收入的和相加;

(198) 将第 100 笔收入输入给计算机;

(199) 将它和前 99 笔收入之和相加;

(200) 打印出 100 笔收入的总和.

显然，这个算法虽是可实现的，但并不是好的算法. 如果收入 1000 笔，算法就会写得更长. 如果事先不知道收入多少笔，只要求累加完当天全部账目就结束，又该怎么写呢？可以让计算机重复执行某一操作，而计算机执行“循环”的操作是轻而易举的. 算法可写为：

（1）设一“计数变量”N，使 N 的初值为零，即 N = 0；

（2）设“累加变量”T，初值为零（T = 0）；

（3）输入一个数给“收入变量”A；

（4）将 A 和 T 的值相加，和放在变量 T 中，即 A + T⇒T；

（5）使 N 的值加 1，即 N + 1⇒N（N 的值表示已累加的数据的个数）；

（6）若 N < 100，则返回（3）继续执行，否则执行（7）；

（7）打印出总和 T 的值.

这个算法比上一个简单明确，如果收入不是 100 笔而是 1000 笔，只需将（6）中的 N < 100 改为 N < 1000 即可. 算法中变量 T 的值是不断改变的，在每次循环中以一个新的值（A + T 的原值）代替它的原值，然后再以 T 的新值作为下一次运算的基础再，求出 T 的下一次的值，如此一次次地求下去直到达到要求为止. 这种方法称为“迭代”. T 称为迭代变量. 计算机算法的最大特点就是“迭代”. 利用计算机高速运算的特点，执行多次循环，通过“迭代”实现各种运算.

例 **2**　求 $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{100}$ 的值. 可以写出下面的算法.

（1）使 S = 0（S 作为累加变量）；

（2）使 N = 1（N 代表分母）；

（3）$S+\frac{1}{N}\Rightarrow S$；

（4）$N+1 \Longrightarrow N$；

（5）若 N≤100，转去执行（3）以及其后的各步骤；否则执行（6）；

（6）打印 S 的值（即所求之总和）.

其中（3）是执行“迭代”的操作，S 是迭代变量. 请读者分析：第（5）步中 N≤100 的含义是什么？能否写成“N < 100”？为什么前一个例子（例 1）的第（6）步用“N < 100”而本例中用“N≤100”？

写出算法之后，再根据它写出用某种高级语言表示的程序. 程序设计的关键在于设计出一个好的算法. 一个好的算法应当是：能够获得正确的结果；容易阅读理解，即易读性好；计算机执行时具有较高的效率（指得到结果所花的时间较少和占计算机内存量较少）.

算法应当具有以下几个特性：

（1）有穷性. 一个算法应当在执行有穷步之后结束. 不应当包含无终止的循环.

（2）确定性. 算法中的每一个步骤，必须是确切定义的，而不应当含混不清或模棱两可的.

（3）具有零个或多个输入量，即在算法开始执行前对算法最初给出的量.

（4）算法执行完毕后有一个或多个输出量. 例如求 m 和 n 的最大公约数 r. m 和 n 为两个输

入量，r 为输出量.

（5）可执行性. 算法中的每一步都是能够准确地进行的. 如，进行 B/A 的除法运算，当 A 等于零时，是无法执行的.

算法贯穿于程序设计的始终，希望读者对算法给予足够的注意. 在拿到一个问题之后应当首先构造出一个好的算法. 在本书各章中都贯彻这一原则.

一个算法可以用自然语言来表示，也可以用流程图来表示（见下节），还可以用介于自然语言和高级语言之间的一种称为伪代码（pseudo code）来表示. 用计算机语言（包括低级语言和高级语言）写的程序也是算法的表示形式.

§1.5 流 程 图

所谓流程图就是用某种形式的图来表示一个算法. 常用的流程图是由一些具有特定含义的框和流向线组成的.

最常用的流程图符号见图 1.9.

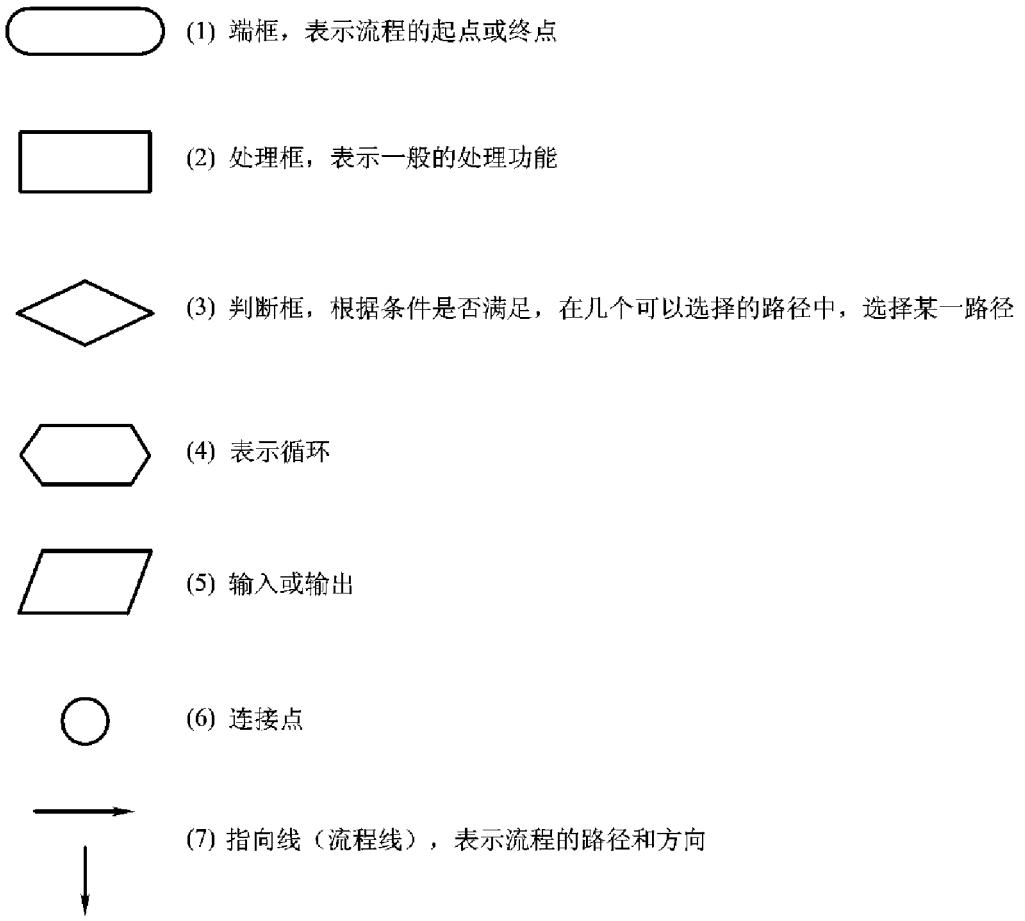


图 1.9 流程图符号

可以用流程图表示上一节中例 1 和例 2 的算法，见图 1.10 和图 1.11.

流程图能形象地表示一个算法，清楚地表示出各个步骤间的先后次序. 但画这种流程图比较麻烦，而且占篇幅. 尤其是处理一个复杂的问题时尤为突出. 但目前这种流程图已被广泛使用，因

此学习计算机课程的读者仍应能看懂和使用它.下一节介绍的结构化流程图被认为更好一些,适合于结构化程序设计.

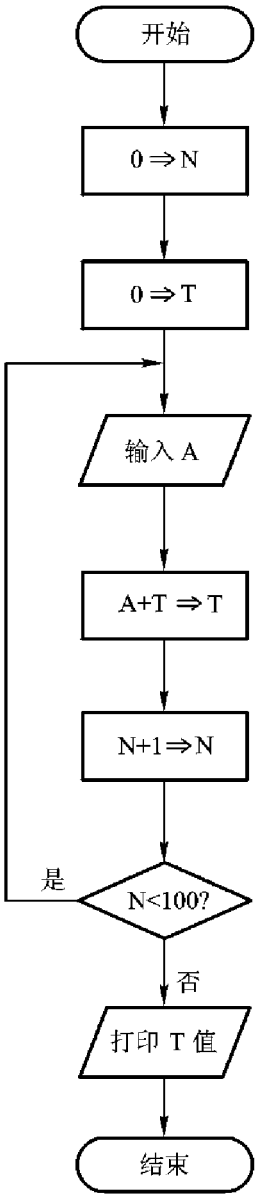


图 1.10

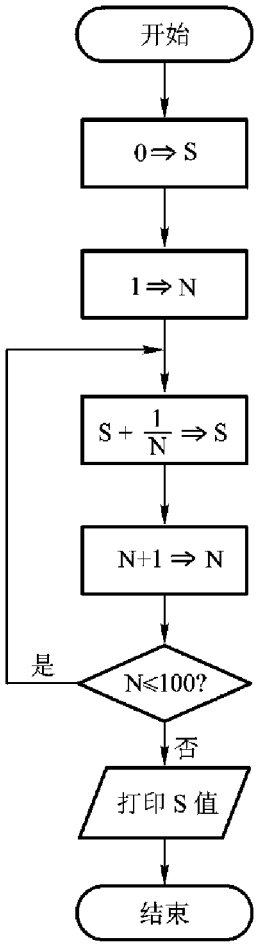


图 1.11

§1.6 结构程序设计和结构流程图

过去一个时期,程序设计基本上是无章可循的,人们可以随心所欲地编写程序,只要符合语言规则并获得正确结果即可.评价一个程序的标准首先看它运行时的效率(运行时间长短和占内存多少).人们往往为了追求时间和内存量微小的节约而千方百计地想出各种小小的"技巧".但是这样做的结果往往使程序的流程转来转去,使人难以看懂程序和理解算法的思路,使得软件的维护费用日益增高.另一方面由于计算机科学技术的迅速发展,内存量和速度都大大提高了,而硬件的价格却不断下降.因而不必为了节省微小的速度和内存量而采取使程序晦涩难懂的小技巧,牺牲了程序的易读性.现在评价算法和程序的标准,除了算法正确这一不言而喻的前提外,

首先是程序结构清楚，易读性好，其次才是效率. 归根结底，只有程序具有良好的结构，易于设计和维护，减少软件成本，从整体来说才是真正的高效率.

荷兰学者 **Dijkstra** 提出了“结构化程序设计”的思想，它规定了一套方法，使程序具有合理的结构，以保证和验证程序的正确性. 这种方法要求程序设计者不能随心所欲地编写程序，而要按照一定的结构形式来设计和编写程序. 它的一个重要目的是使程序具有良好的结构，使程序易于设计、易于理解、易于调试修改，以提高设计和维护程序工作的效率.

结构化程序规定了以下三种基本结构作为程序的基本单元.

（1）顺序结构　它是最简单、最基本的一种结构. 在这个结构中的各块是按顺序执行的. 见图 1.12. 每一块可以包含一条或若干条可执行的指令.

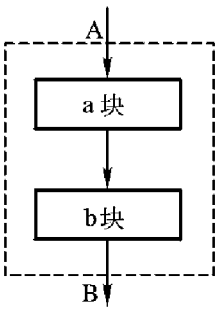


图 1.12

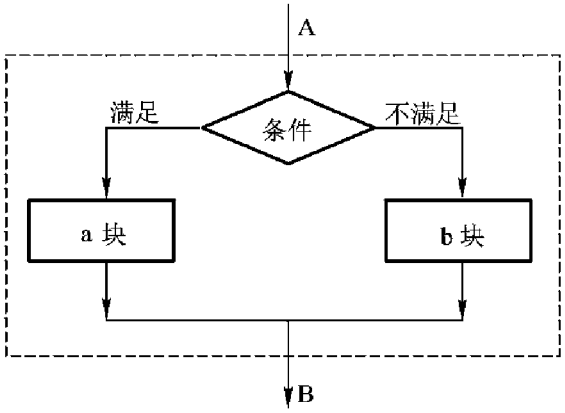


图 1.13

（2）判断选择结构. 见图 1.13.

根据给定的条件是否满足执行 a 块或 b 块.

（3）循环结构. 见图 1.14 和图 1.15.

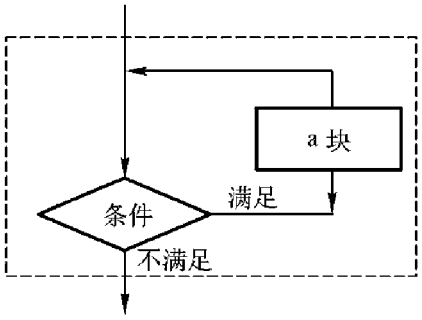


图 1.14

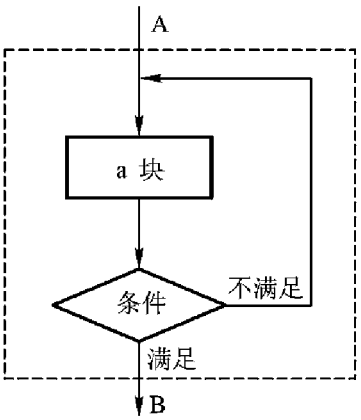


图 1.15

图 1.14 表示的结构称为“当型”循环. 当给定的条件满足时执行 a 块，否则不执行 a 块而执行虚线框下面的部分. 图 1.15 表示的结构称为“直到型”循环，它的含义是：执行 a 块直到满足给定的条件为止（满足了条件就不再执行 a 块）. 这两种循环的区别是：当型循环是先判断（条件）再执行，而直到型循环是先执行后判断.

以上三种基本结构可以派生出其他形式的结构. 由这三种基本结构所构成的算法可以处理任何复杂的问题. 所谓结构化程序就是由这三种基本结构所组成的程序.

可以看到，三种基本结构都具有以下特点：

（1）有一个入口.

（2）有一个出口.

（3）结构中每一部分都应当有被执行到的机会，也就是说，每一部分都应当有一条从入口到出口的路径通过它（至少通过一次）.

（4）没有死循环（无终止的循环）.

结构化程序要求每一基本结构具有单入口和单出口的性质是十分重要的，这是为了便于保证和验证程序的正确性. 设计程序时一个结构一个结构地顺序写下来，整个程序结构如同一串珠子一样顺序清楚、层次分明. 在需要修改程序时，可以将某一基本结构单独孤立出来进行修改，由于单入口单出口的性质，不致影响到其他的基本结构.

在结构化程序设计中，使用一种“结构化流程图”. 它的基本成分有以下三种：

（1）一条简单的指令，用一个矩形框来表示，见图 1.16.

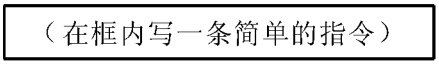


图 1.16

顺序结构可以表示如图 1.17.

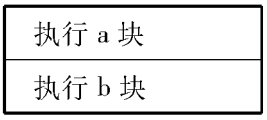


图 1.17

（2）判断选择结构用图 1.18 形式的框图表示：

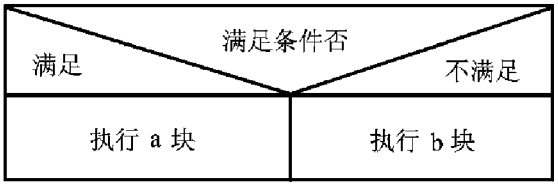


图 1.18

（3）循环结构用图 1.19 和图 1.20 形式的框图表示. 图 1.19 表示的是当型循环，图 1.20 表示的是直到型循环.

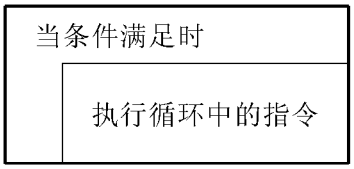


图 1.19

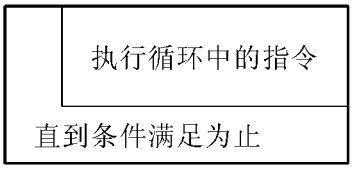


图 1.20

§1.4 中例 1 的算法可用结构流程图来表示，见图 1.21. 例 1 算法中的第（3）到第（6）步构成一个直到型循环，终止循环的条件是 $N \geqslant 100$（因为如果 $N < 100$，还要继续执行循环）.

例 2 算法用结构流程图表示如图 1.22. 也可以将算法改写为：

（1）S = 0；

（2）N = 1；

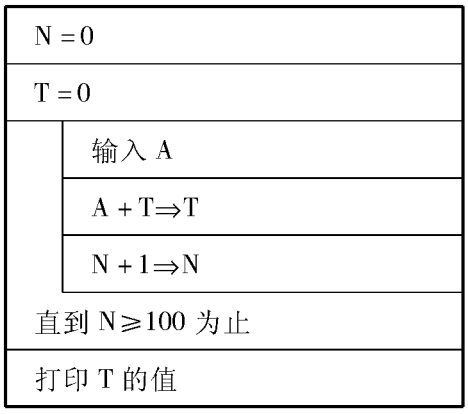


图 1.21

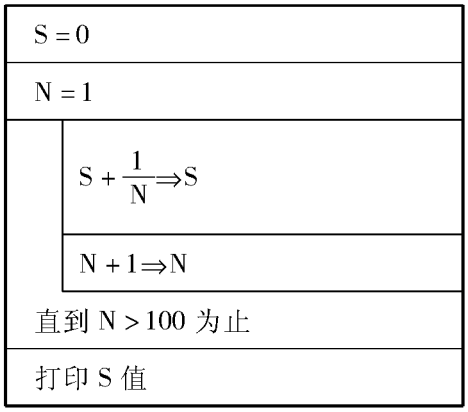


图 1.22

（3）若 N≤100，则执行（4）到（6），否则转到（7）；

（4）$S+\frac{1}{N}\Longrightarrow S$；

（5）$N+1\Longrightarrow N$；

（6）转回（3）；

（7）打印 S 值.

其中（3）到（6）步构成一个当型循环，用结构化流程图表示如图 1.23.

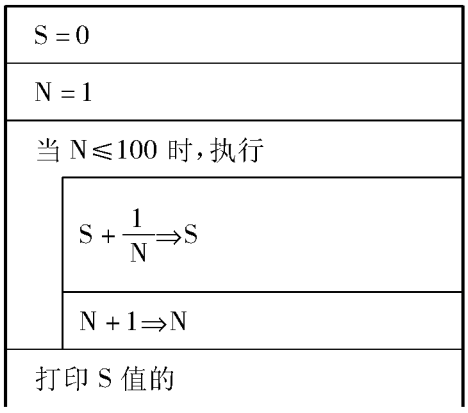


图 1.23

可见一个问题可以有不止一个算法，可以根据要求和各人的习惯选择合适的算法.

例 **1**　求 $1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$（直到第 100 项）的值. 判断第 100 项的绝对值是否大于 0.005，并打印出相应的信息.

可以直接画出结构化流程图，见图 1.24.

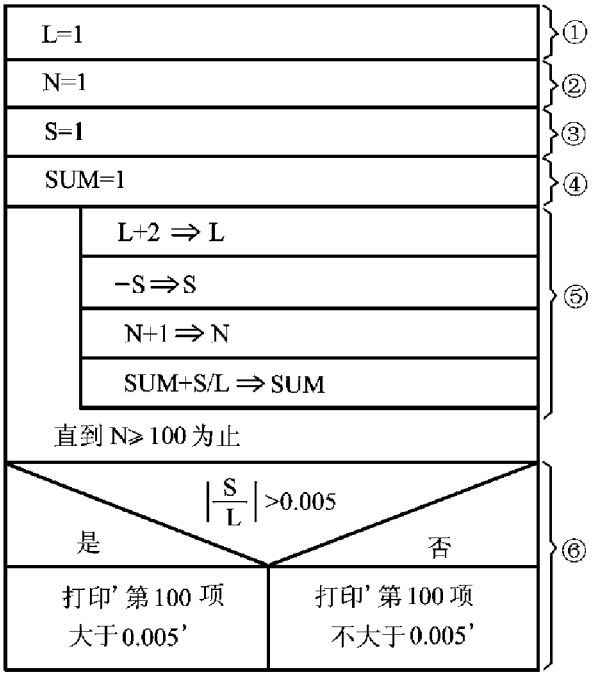


图 1.24

其中 SUM 代表累加和，在开始时使它的初值为 1，然后每次在其上增加下一项的值. I 代表分母的值，使其初值为 1. N 是作为计数用的，它代表已累加了多少项. S 代表符号，它的初值为 1，以后轮流取 1 和 −1，以控制各项符号的变化.

在执行此算法时，各变量值的变化情况如下表所示：

第 n 次循环	L 原值	S 原值	N 原值	L 新值	S 新值	N 新值	SUM 值	是否继续执行循环
执行循环前	1	1	1				1	
第 1 次	1	1	1	3	−1	2	$1-\frac{1}{3}$	执行
第 2 次	3	−1	2	5	1	3	$1-\frac{1}{3}+\frac{1}{5}$	执行
第 3 次	5	1	3	7	−1	4	$1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}$	执行
第 4 次	7	−1	4	9	1	5	$1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}$	执行
第 99 次	197	1	99	199	−1	100	$1-\frac{1}{3}+\frac{1}{5}-\frac{1}{199}$	不执行

可以看出，上一次循环中的 L，S，N 的新值就是下一循环中 L，S，N 的原值. 每一次循环它们的值都在改变，这就是“迭代”，SUM 也进行迭代累加. 请注意，在循环开始时 SUM 的值为 1，即已有了多项式第一项的值，因此只应再加 99 项，因此执行循环 99 次而不是 100 次. 停止循环的条件是 N≥100 而不是 N＞100，这是因为在加完第 100 项（第 99 次循环）后 N 的值为 100，此时不应再执行循环了.

最后一次循环执行完后的 S/L 就是第 100 项的值，判断它的绝对值是否大于 0.005，然后打印相应的信息.

这个问题的算法是简单的，然而是典型的计算机算法. 希望读者对它能深入地理解和掌握.

图 1.24 流程图可以分成六个基本的单元，其中①、②、③、④为简单的指令，⑤为直到型循环结构，⑥为判断选择结构. 画流程图和写程序时从上到下一个一个地写下来，看流程图和程序时也是从上到下一个结构一个结构地看下来，不会有上下来回跳转. 如同看一本书一样，一段一段看下来，来龙去脉清楚. 这就是结构化程序最突出的优点.

由于结构化流程图是一个结构一个结构顺序组成的，因此它不需要像传统的流程图那样的指向线（流程线）. 所以结构流程图画起来紧凑，占篇幅少，逻辑清楚，容易理解. 在结构化程序设计中得到广泛的应用. 希望读者能熟练地使用它.

结构化流程图是美国 I. Nassi 和 B. Schneiderman 两人在 1973 年提出的方法的基础上形成的，因此又称为 N－S 图.

画出结构流程图后，再根据它编写程序就不难了. 本书第二章开始介绍如何用 FORTRAN 语言来表达和实现一个算法.

考虑到结构化程序设计方法和结构流程图的显著优点以及它们已被日益广泛地使用，因此在本书各章介绍程序设计时全部按照结构化程序设计原则并采用结构流程图.

§1.7　利用计算机解题的全过程

程序设计应当包括从接受任务、分析问题开始，到最后通过计算机运行得到正确结果的全过程. 它一般包括以下几个步骤：

（1）明确所需解决的问题. 必须在接受任务时就确切地弄清楚问题的性质、任务和要求. 例如，给出什么数据，要求得到什么结果，打印格式有什么要求等等. 这一步看起来虽然简单，然而却是很重要的. 如果提出任务和接受任务不是同一个人，则往往需要反复磋商多次才能最后确定. 譬如，对误差的要求，甲方往往希望误差愈小愈好，甚至会提出不切实际的要求，而乙方则往往希望问题简化些. 考虑到所需付出的人力和物力，二者有时是有矛盾的，需要协商找出一个双方都能满意的方案.

还要分析任务的性质和规模，需不需要计算机和应该用什么计算机解决，要考虑到实际的效益.

有的人往往拿到题目不加深入思索就动手编程序，到最后才发现未能满足要求而不得不多次返工修改，这是从事程序设计者所切忌的.

（2）分析问题、构造模型. 把自然界和社会生活中的现象经过一定的简化，用数学语言描述它，例如把一个物理问题用一个联立方程来表示. 这工作叫做建立数学模型.

（3）选择适当的计算方法. 在建立数学模型后，还要找出用计算机解决该问题的近似方法. 例如，求一个函数的定积分，用计算机解决此问题不同于纯数学的方法. 计算机是通过数值计算而得到一个数值解，而它往往只能是近似解. 求定积分可以用矩形法（将曲线梯形面积近似地认为是若干个小矩形之和）、梯形法（认为是若干小梯形之和），也可以用辛甫生法（用抛物线代替梯形的斜边）. 每一种方法的近似程度不同，所花费的计算机时间不同. 应当根据所解决问题的要求选择合适的方法. 研究数值计算方法的学科称为“计算方法”（Computational Method）. 典型的各种数值计算问题都有现成的计算方法可供参考.

（4）确定数据结构和算法. 在选择好的计算方法后，要考虑数据的组织形式，即数据结构还要考虑好用计算机运算的全部步骤. 这一步是人们整理思路的过程，是十分重要的. 如果算法不合适就会导致结果出错或影响效率.

（5）根据已确定的算法画出流程图（结构流程图）. 只要算法和流程图是正确的，编写程序一般不会出现大的逻辑性的错误.

（6）根据流程图编写程序. 这一步又称编码（Coding），将结构流程图中各基本结构代以具体的高级语言（例如 FORTRAN77 语言）即可.

（7）将写好的高级语言源程序穿成卡片，输入计算机，或通过终端将源程序输入计算机. 经过计算机对源程序编译、联接得到计算机能执行的目的程序. 关于对源程序进行编译和联接的方法请见第二章.

（8）进行试算. 一个程序（尤其是一个复杂的程序）往往不是一次就能成功的，需要多次试算并根据试算结果验证程序的正确性，如果程序有错，应检查出错误所在之处并修改程序. 试算时可以用“试验数据”，即简单的、容易验证结果正确性的数据. 这一步骤称为“调试”（Debugging）.

（9）通过调试得到可供正式运行的程序后，上机正式运行，送入正式数据，得到必要的运算结果.

（10）整理资料，写出报告. 检查所得结果是否全部满足预定的要求. 如果符合要求，则应当整理结果写出程序说明书. 它包括：题目；任务要求；原始数据；计算方法；算法或流程图；程序清单；运行打印结果；操作说明及设备配置等. 一个好的有使用价值的程序往往不是只用一次就完了的，它可以多次使用或提供其他人使用，有了程序说明书才能帮助别人理解和使用你的程序. 当然对一些小的程序可以不必作这样的要求，但是即便是初学者也应当养成保存资料和分析结果的良好习惯.

计算机解题的整个过程见上页图 1.25.

习　题

1. 请用你自己的语言叙述什么叫电子计算机.

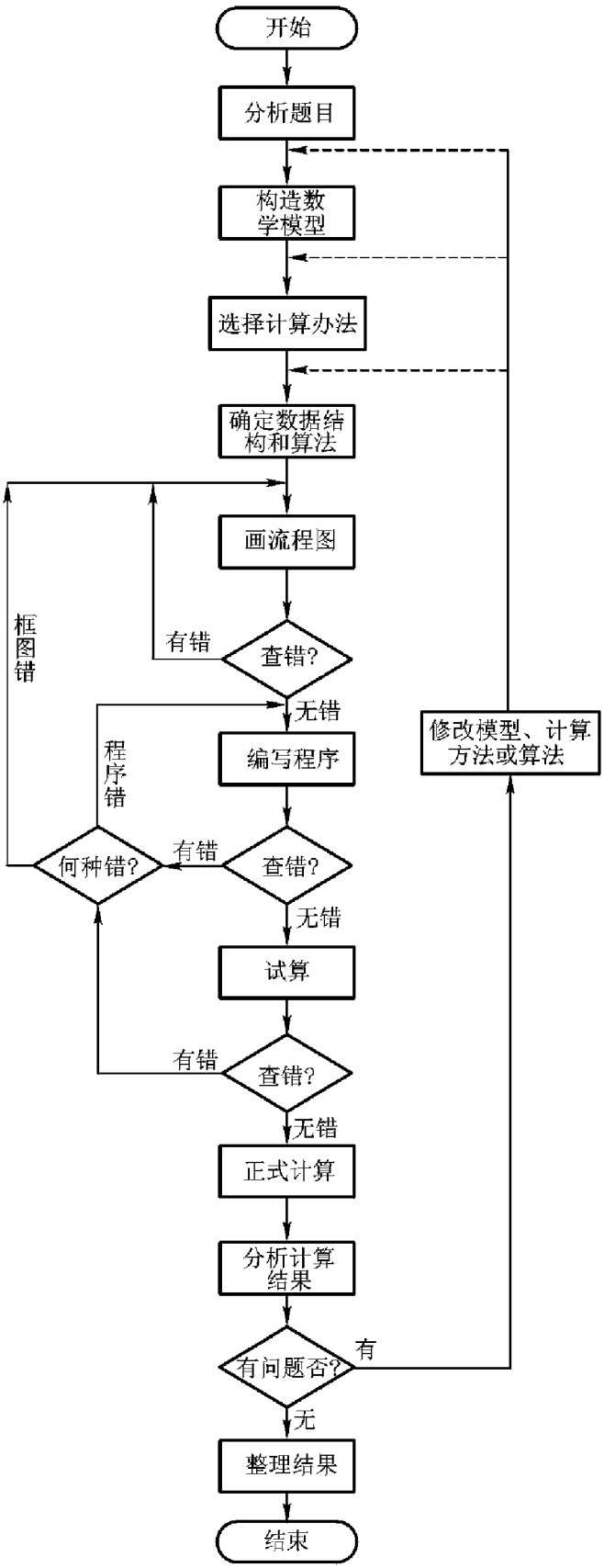

开始
分析题目
构造数学模型
选择计算办法
确定数据结构和算法
画流程图
查错?
有错
框图错
无错
编写程序
程序错
何种错?
有错
查错?
无错
试算
有错
查错?
无错
正式计算
分析计算结果
有问题否?
有
修改模型、计算方法或算法
无
整理结果
结束

图 1.25

2. 电子计算机由哪些部分组成.

3. 什么叫低级语言,什么叫高级语言?什么叫高级语言编译程序?

4. 什么叫结构化程序设计?它有什么优点?

5. 请画出结构化程序的三种基本结构(顺序结构、判断选择结构、循环结构)的结构化流程图.

6. 求 $\sum_{n=1}^{1000} n$ 之值. 请写出算法,并画出结构流程图.

7. 求 $2^0+2^1+2^2+\cdots+2^{50}$ 之值. 请写出算法,并画出结构流程图.

8. 求 $\frac{1}{1\times2}+\frac{1}{2\times3}+\frac{1}{3\times4}+\cdots+\frac{1}{n(n+1)}$ 之值,$n=20$. 请写出算法,并画出结构流程图.

9. 用公式 $\frac{\pi}{2}=\frac{2\times2}{1\times3}\times\frac{4\times4}{3\times5}\times\frac{0.6}{5\times7}\times\cdots\times\frac{2n\times2n}{(2n-1)\times(2n+1)}$,求 π 的近似值,$n=50$. 请写出算法,并画出结构流程图.

10. 求 $n!$ 之值. 请写出算法,并画出结构流程图. n 的值由键盘输入计算机.

第二章　FORTRAN 语言的基本知识

§2.1　FORTRAN 语言简介

FORTRAN 语言是最早出现并得到广泛使用的一种计算机语言. 它是 FORmula TRANslation（公式翻译）的缩写，最初是为数值计算而设计出来的. 第一个 FORTRAN 文本是 1954 年提出的，1956 年开始真正得到使用. 三十年来形成了许多版本，其中最流行的是 FORTRAN Ⅱ（1958 年）和 FORTRAN Ⅳ（1962 年）. 1966 年美国国家标准化协会（Americian National standard Institute. 简称 ANSI）以 FORTRAN Ⅳ为基础制订了美国标准文本，即 ANSI X 3.10－1966 FORTRAN，简称 FORTRAN 66. 实际上 FORTRAN 66 标准就是标准 FORTRAN Ⅳ. 由于 FORTRAN 语言在世界范围内迅速地得到推广，1972 年国际标准化组织（International Standard Organization，简称 ISO）宣布将 FORTRAN 66 作为 ISO 的 FORTRAN 标准文本. 至今大量的应用程序是用 FORTRAN 语言写的.

近二十年来，随着计算机应用的发展，原有的 FORTRAN Ⅳ已不能满足实际使用的要求，不少厂家都对 FORTRAN Ⅳ进行了扩充. 实际上目前使用的 FORTRAN Ⅳ都在不同方面、不同程度上扩充了其功能. 为了适应发展的需要，在各计算机厂商扩充 FORTRAN Ⅳ的基础上，美国国家标准化协化（ANSI）于 1976 年提出了修订文本，征求意见，并于 1978 年正式公布为新的国家标准文本，即 ANSI X 3.9－1978 FORTRAN，为了与原标准（FORTRAN 66）相区别，新标准被称为 FORTRAN 77.

FORTRAN 77 对 FORTRAN 66 在许多方面作了重要的改进. 最早的 FORTRAN 是专为数值计算设计的，不适用于其他的数据处理. FORTRAN 77 增加了字符处理功能，使 FORTRAN 能应用于非数值运算领域. FORTRAN 77 还增加了“块 IF”语句，便于实现判断选择结构和循环结构，这就使它能方便地用于结构化程序设计. 此外，FORTRAN 77 还增强了输入输出的功能和文件处理能力，对原 FORTRAN 标准中许多部分作了改进（如允许不同类型的量混合运算；数组下界可以等于或小于零；下标表达式可以为任意的整型表达式；DATA 语句中可以用隐循环等），使 FORTRAN 77 成为功能较强的一种计算机语言. 它既可用于数值计算（这仍然是 FORTRAN 主要的应用领域），又可用于一般数据处理（例如管理、计算机辅助设计等）.

考虑到 FORTRAN Ⅳ已在世界范围内广泛流行，大量的 FORTRAN Ⅳ源程序和程序库为千千万万的人每日每时所使用，FORTRAN 77 基本上是与 FORTRAN 66 标准兼容的，即用 FORTRAN 66（即 FORTRAN Ⅳ）编写的源程序可以不加修改（或只作很小的修改）即能在 FORTRAN 77 编译系统支持下运行. FORTRAN 66 中有些语句不符合结构化原则，在结构化程序设计中已不被使用，但是在 FORTRAN 77 中仍予保留（但不提倡使用），这就不致使二十年来积千万人心

血的应用程序作废，保持了 FORTRAN 语言的继承性.

FORTRAN 77 包括一个全集和子集，一般微型计算机上配置的 FORTRAN 77 不是全集而是子集，不具备 FORTRAN 77 的某些功能. 请读者使用计算机前了解所用计算机的 FORTRAN 77 的功能.

FORTRAN 77 已在世界各国广泛使用，在我国许多计算机系统上也已配备了 FORTRAN 77 编译程序，随着结构化程序设计方法的推广，FORTRAN 77 将在我国得到迅速的普及使用. 按照结构化的原则用 FORTRAN 77 语言编写新的程序会大大提高程序设计的效率和质量. 不符合结构化原则的原 FORTRAN 66 的语句虽然为 FORTRAN 77 所保留，但今后使用它们的机会愈来愈小，故本书只在第十一章中作简单介绍，以使读者在阅读 FORTRAN Ⅳ程序时不致感到困惑.

§2.2 几个简单的 FORTRAN 77 程序

让我们先看几个简单的 FORTRAN 77 程序，然后再对它们作初步的分析，以建立对 FORTRAN 的初步知识.

例 1 给出半径 R，高度 H，写出求圆的周长、圆面积、圆柱全面积、圆锥侧面积、圆锥体积的程序.

从数学公式中知道：

圆周长 $C = 2\pi R$

圆面积 $S1 = \pi R^2$

圆柱全面积 $S2 = 2\pi R(H + R)$

圆锥侧面积 $S3 = \pi R\sqrt{R^2 + H^2}$

圆锥体积 $V = \frac{1}{3}\pi R^2 H$

如果 R = 10 米，H = 8 米，则程序可编写如下：

```
*   EXAMPLE 1
      PI = 3.14159265
      R = 10.0
      H = 8.0
      C = 2 * PI * R
      S1 = PI * R * R
      S2 = 2 * PI * R * (H + R)
      S3 = PI * R * SQRT(R * R + H * H)
      V = (PI * R * R * H)/3
      PRINT *,'C = ',C
      PRINT *,'S1 = ',S1
      PRINT *,'S2 = ',S2
      PRINT *,'S3 = ',S3
      PRINT *,'V = ',V
      END
```

图 2.1

下面对程序先做些粗略的说明，读者只需对它有初步的了解即可，关于程序中各语句的作用

和规则在以后几章中将会有详细的介绍.

程序中的第一行是注解行. 凡在本行第一列处写上“ * ”号或字母 C 的,该行作为注解行. 在注释行中可以写入任意需要的内容,对程序运行不起影响.

第二行到第九行是“赋值语句”,其作用是将“等号”右边的值送到“等号”左边的变量中去. 第二行以变量 PI 代表 π,它的值定为 3.14159265. 第三、四行分别将已知的参数放到 R 和 H 中. 第五行到第九行的作用是根据已知公式运算后得到所需的值(C,S1,S2,S3,V),其中第八行中的 SQRT 是平方根函数的符号(SQRT 是英文 SQuare RooT 平方根的缩写),SQRT(R * R + H * H)的作用是计算$\sqrt{R^2+H^2}$. 第十行到第十四行是打印输出的部分,PRINT 是“打印输出”的意思. PRINT 后面的星号(*)表示按计算机事先规定好的格式打印出所指定的信息. PRINT 语句中引号内的内容(如'C = '中的 C =)在打印时原样照印. 然后接着打印其他项的值(如 C 的值). 第十五行 END 表示程序结束.

程序运行时打印出以下信息:

C = 62.8318600

S1 = 314.1593000

S2 = 1130.9730000

S3 = 2513.2740000

V = 837.7581000

例 **2** 输入 100 个数,将其相加,打印出它的和. 这个问题的算法和流程图已在第一章中给出(见第一章 §1.4 节和图 1.10、图 1.21). 现在根据图 1.10 和图 1.21 写出 FORTRAN 77 程序.

```
*    EXAMPLE 2
          PROGRAM EXAM2
          N = 0
          T = 0
    10    READ *,A
          T = A + T
          N = N + 1
          IF(N.LT.100) GOTO 10
          PRINT *,T
          END
```

图 2.2

这个程序比上一程序多了一些内容. 第二行以 PROGRAN 开头,它是“程序语句”,作用是为本程序定一个名字,今程序名定为 EXAM2. 程序语句是程序中可有可无的语句,写上此语句为程序起个名字以便查找方便. 第三、四行是赋值语句,使 N 和 T 的初值为零. 第五行 READ 是“读入”语句,从计算机的输入设备(如键盘)输入一个数据送给变量 A,星号“ * ”表示用自由格式(计算机规定的一种比较方便的输入数据的方式)输入数据. 第五行上第一、二列有数字“10”,它称为标号(Label),或称行标号,它是一行的标志,当需要从程序中其他部分转来本行继续执行时,就用此标号作标志. 第六行是将 T 的原值加上 A 的值再将和送回 T 中. 第七行的作用将 N 的原值加 1 再送回 N,即使 N 的值加 1. 第八行是一个条件判断语句,括号内(N.LT.100)是一个给

定的条件，表示“N 小于 100”，“.LT.”是比较符号（LT 是 Less Than 的缩写），本行语句的意思是：“如果满足 N＜100 这个条件，则转到标号为 10 的行去，如果不满足此条件则继续执行其下一行语句.”这个 IF 语句用来控制循环的次数，使第五行到第七行的语句重复执行 100 次. 第八行 PRINT 语句打印出 100 个数的累加和. 然后结束.

例 **3**　求 $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{100}$ 的值. 其算法见 §1.4 节，流程图见图 1.11 和图 1.22. 根据流程图可以写出 FORTRAN 77 程序如下：

```
*   EXAMPLE 3
        PROGRAM EXAM3
        S=0
        N=1
   10   S=S+1./N
        N=N+1
        IF(N.LE.100) GOTO 10
        WRITE(*,*)'S=',S
        END
```

程序中 WRITE(*,*)的作用和上面二例中的 PRINT * 相同，实现打印输出. FORTRAN 子集不包括 PRINT 语句，而只能用 WRITE 语句.

第七行 IF 语句中“LE”的含义是“小于或等于”，是英文“Less than or Equal to”的缩写. IF 语句的作用是：如果“N≤100”这一条件满足，则转去标号为 10 的行去，以实现循环 100 次. 对此程序的其他部分不作过多解释，请读者自己看懂本程序.

程序运行结果如下：

```
S=5.1873780
```

例 **4**　求 $1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$ 直到第 100 项的值. 判断第 100 项的绝对值是否大于 0.005，并打印出相应的信息. 算法已在 §1.6 节介绍，流程图见图 1.24.

根据流程图写出 FORTRAN 程序如下：

```
*   EXAMPLE 4
        PROGRAM EXAM4
        L=1
        N=1
        S=1.0
        SUM=1.0
    1   L=L+2
        S=-S
        N=N+1
        SUM=SUM+S/L
```

```
      IF(N.LT.100) GOTO 1
      IF(ABS(S/L).GT.0.005) THEN
          WRITE(*,*)'GRATER THAN 0.005'
      ELSE
      WRITE(*,*)'LESS THAN OR EQUAL TO 0.005'
      END IF
          WRITE(*,*)'SUM =',SUM
      END
```

运行结果如下：

```
GRATER THAN 0.005
SUM =7.828981E-001
```

多项式的和为0.7828981.

请读者对照图1.24看懂这个程序.从第十二行到第十六行是一个“块IF”,它以块IF语句(IF…THEN语句)开始,以END IF语句结束.它的含义是:“若S/L的绝对值大于0.005,则打印‘第100项大于0.005,’否则,打印‘第100项小于或等于0.005’”.它是一个判断选择结构.第十二行中的ABS是绝对值函数(ABS是英文ABSolute value(绝对值)的缩写).“GT”是“大于”(Greater Than的缩写).关于“块IF”的详细介绍请看第五章.

例**5**　求5!,6!和7!.

先画出求n!的结构流程图.见图2.3.

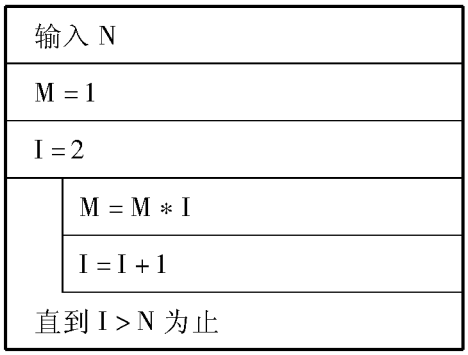


图2.3

由于需要求三个数的阶乘,如果重复写出三段内容重复(而仅仅是具体数据N不同)的程序,显然是不必要的.FORTRAN提供了子程序的功能,将完成同一操作的语句编成一个子程序,可以多次调用该子程序完成同一类操作,可以编一个主程序和一个子程序来实现题目的要求.

(1)主程序:

```
*   MAIN PROGRAM
      PROGRAM EXAM5
      K = M(5)
      J = M(6)
```

```
      L = M(7)
      WRITE ( * , * ) K,J,L
      END
```

图 2.4

（2）子程序

```
*   FUNCTION SUBPROGRAM
      FUNCTION M(N)
      M = 1
      I = 2
 10   M = M * I
      I = I + 1
      IF (I. LE. N) GOTO 10
      END
```

图 2.5

程序由两部分组成：一个主程序（名为 EXAM5），一个子程序，该子程序是函数子程序，在主程序和子程序的第一行分别用注解行对程序的性质作了说明. 子程序中第二行为函数子程序的定义语句，它指定这个函数子程序的名字为 M，M 后面括弧内有一变量 N，它的值需要在调用此子程序时由主程序赋给它. N 有了值以后就能通过子程序中的语句算出 N!，例如 N = 5，子程序就求出 5!，这个结果存放在 M 中，并带回到主程序. 子程序也是以 END 语句结束.

主程序的第三行 K = M(5)，是调用函数子程序 M，将 5 代入子程序中的 N，这样就通过调用子程序求出了 5!，并将 5！ 的值赋给变量 K. 同样第四行的作用是求出 6！ 并将结果赋给 J. 第五行的作用是求出 7！ 并将结果赋给 L. 第六行 WRITE 语句打印出 K，J，L 的值，也就是 5!，6！ 和 7！ 的值.

程序运行结果如下：

```
120      720      5040
```

对以上几个程序只要求有一个大致的了解即可，有关内容将在以后几章中详细介绍.

§2.3 FORTRAN 程序的构成

通过上面几个程序例子，可以知道一个 FORTRAN 程序是如何构成的，程序中包括哪些成分.

1. 一个 FORTRAN 程序可以只包括一个主程序，也可以由一个主程序和若干个子程序组成. 每一个主程序和每一个子程序分别都作为一个程序单位. 所以，一个 FORTRAN 程序由一个或若干个程序单位组成.

每一个程序单位分别独立地编写、输入（计算机）和编译，把它们分别翻译成由机器指令组成的目的程序，分别保存在磁带或磁盘上. 然后将一个程序的各程序单位联接起来成为一个整体，就可以运行了.

把实现不同功能的部分程序分别编成子程序，需要时像搭积木似地将有关程序单位组成一

个程序，这样可以提高程序设计的效率，增加程序使用的通用性.

2. 每一个程序单位以 END 语句作为结束标志. 也就是说，一个程序最后一个语句必须是“END”. 一个程序单位必须有而且只能有一个 END 语句. 在 FORTRAN 77 中 END 既作为程序单位结束标志，又作为“程序停止运行”的指令（在主程序中）或“返回主程序”的指令（在子程序中）. 而在 FORTRAN 66 中，END 只作程序单位结束标志而不作为可执行语句，由 STOP 语句停止程序运行，用 RETURN 语句使流程从子程序返回主程序. 如果读者使用的是 FORTRAN Ⅳ编译系统，则应在主程序的 END 前一行加一个 STOP 语句，在子程序的 END 前面加一个 RETURN 语句.

3. 每一个程序单位由若干行组成. FORTRAN 中的行有两类：

（1）语句行. 由一个 FORTRAN 语句组成. 例如，赋值语句、读语句、打印语句等都是 FORTRAN 语句. 它们对程序的执行有影响.

（2）非语句行. 对程序的执行不起影响，如注解行，它们不被翻译成机器指令，只是在列程序清单时照样打印出来. 一个程序中可以有任意个注释行（也可以没有），但一个程序单位不能只由注解行组成. 注解行的作用是使人们能对程序（或程序的一部分）作必要的注解或说明，以帮助别人理解程序或供备忘.

FORTRAN 程序的基本成分是语句. FORTRAN 规定，一行内只允许写一个语句，不能在一行内写几个语句. 如果一个语句很长，一行内写不下，可以写在连续的几行内（用续行标志）.

4. FORTRAN 语句可以有标号，也可以没有标号. 是否设标号根据需要而定. 标号的作用是标志一个语句以便引用. 同一程序单位中不能有一个以上相同标号的语句. 标号大小不影响语句执行的顺序. 程序执行顺序由语句出现的次序决定.

标号的范围是 1 位整数到 5 位整数（1 到 99999），不能带正负符号.

5. 一个程序单位中各类语句的位置有一定的规定，除了 END 语句必须出现在最后一行外，PROGRAM 语句必须出现在主程序的第一行（PROGRAM 语句只能出现在主程序中而不能出现在子程序中. 主程序中也可以没有 PROGRAM 语句，但如果写 PROGRAM 语句的话，必须在第一行）. 子程序的第一个语句必须是子程序定义语句（FUNCTION 语句、SUBROUTINE 语句或 BLOCK DATA 语句）.

程序单位中各语句顺序请见附录Ⅳ.

§2.4 FORTRAN 源程序的书写格式

FORTRAN 源程序必须严格地按照规定的格式书写. 用标准的 FORTRAN 程序纸写的程序见图 2.6.

相信读者自己能看懂这个程序.

一张 FORTRAN 程序纸一般分为二十行，每一行有八十列，即一行内有八十个格子，每个格子内只能写一个字符. 这八十列又可分为四个区，分别书写源程序中不同部分的内容.

1. 标号区. 从第 1 列到第 5 列. 它可以是空白无标号. 如果语句设标号的话必须写在这个区

FORTRAN 程序纸　　第　页 / 共　页

计算机	单 位		设 计		审 核	
	程序名	求一元二次方程式的根	电 话		日 期	
	作业名		备 注			

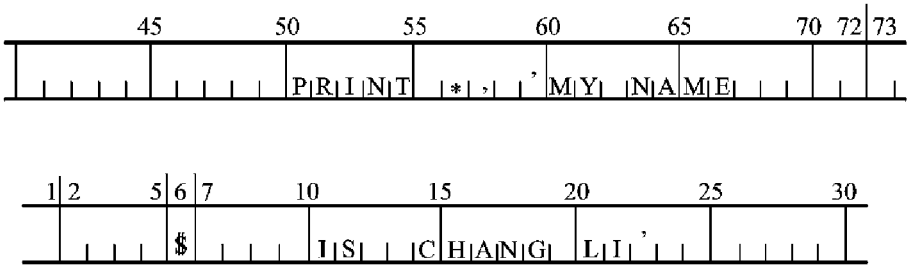
0	STATEMENT NUMBER (2–5)	CONT (6)	FORTRAN STATEMENT (7–72)	IDENTIFICATION (73–80)
*			THEN ROOTS OF QUADRATIC EQUATION	1
			PROGRAM EXAMO	2
			READ *, A, B, C	3
			D = B * B - 4 * A * C	4
			IF (D .GE. 0) THEN	5
			X1 = (-B + SQRT (B * B - 4 * A * C)) / (2 * A)	6
			X2 = (-B - SQRT (B * B - 4 * A * C)) / (2 * A)	7
		$		8
			PRINT *, 'X1 =', X1	9
			PRINT *, 'X2 =', X2	10
			ELSE	11
			PRINT *, 'THERE ARE COMPLEX ROOTS'	12
			END IF	13
			END	14

图 2.6

内. 标号区内的空格不起作用，如 10 ␣␣␣，1 ␣␣␣0，␣␣␣10，␣1␣0␣作用相同（␣代表一个空格字符. 实际上在纸上不出现“␣”符号，它只代表纸上的一个空白位置）.

注解行可以占用标号区，凡第一列为“ * ”或“C”的，该行按注解行处理.

2. 续行标志区. 第 6 列. 在该列上写一个非零非空格的字符（必须是 FORTRAN 字符集中的字符. FORTRAN 字符集见 §2.5），则表示本行是上一行的继续，在编译时会把这两行连接在一起，成为一个语句. 在图 2.6 程序中，第 8 行第 6 列上的“$”符号，表明第 8 行是第 7 行的“继续行”. FORTRAN 标准规定，一个语句最多可以有 19 个继续行，即一个语句最多可包括 20 行，共 1320 个字符. 事实上不必写到一行的最后一列（第 72 列）后才转行，可以从任一列中断而从下一行接着写下去，下一行也不必从第 7 列开始写，而可从第 7 列以后的任一列开始. 图 2.6 程序就是这样的. 在编译时会把上一列最后那些连续的空格和继续行上开头那些空格去掉. 但应注意，如果引号内的字符很多，一行写不完而需转行时，引号内的空格是有效而被保留的. 如：



在打印时将第一行 NAME 字后面的五个空格和继续行第 7 列开始的四个空格保留，打印结果为：

MY NAME　　IS　CHANG LI

续行标志区可以写任意一个非零非空格字符，但是如果在第 6 列上写一个数字“1”，而在续行的第 7 列开始是语句的继续部分“3.14159”，在看程序时很容易把二者混起来而误成“13.14159”. 因此最好用程序中不常用的特殊字符（如“$”）作续行标志.

注解行不能使用继续行. 如果需要注解的内容超过一行，不能在下一行的第 6 列使用续行标志，而应在下一行第一列再用“ * ”或“C”字符再开辟一个注解行. 也就是用多个注解行连续作注解.

3. 语句区. 从第 7 列到 72 列. 但不必一定从第 7 列开始. 为了和续行标志（第 6 列）分隔开以免产生混淆，常常有意将语句从第 8 列或第 9 列开始写（见图 2.6）. 语句区内的空格是不起作用的. 如：COS（X + Y）和 CO␣S␣（X␣+ Y）等价.

4. 注解区. 第 73 列到第 80 列，这是留给程序设计者作标志或注解用的，可以用来写上行的编号（例如用 00100010 表示此行是第 1 页第 10 行），以便按号查找程序中的语句.

§2.5　FORTRAN 字符集

在 FORTRAN 程序中允许使用以下字符：

1. 字母字符. 二十六个大写字母.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

2. 阿拉伯数字. 十个数字.

0 1 2 3 4 5 6 7 8 9

3. 专用字符. 十三个.

专用字符	字符名
	空格
+	加号
-	减号
*	乘号
/	除号
=	等号
(	左括号
)	右括号
,	逗号
.	小数点
$	货币号
'(或')[注]	撇号,引号
:	冒号

以上49个字符是任何一个计算机系统都可以使用的FORTRAN字符,有的计算机系统的FORTRAN编译系统对字符集作了一些扩充. 如有的允许使用小写字母,这是为了适应人们的习惯,但FORTRAN编译程序不区分大写字母和小写字母,如PRINT和print作用是一样的.

§2.6 FORTRAN源程序输入计算机的方式

写好源程序后,要将它送到计算机内. 常用的输入方式有以下几种:

1. 从终端键盘输入

微型计算机系统和采用分时方式的大、中型计算机系统用这种方式. 使用终端输入时,首先调入"文本编辑程序"(每一种计算机系统都有"文本编辑程序",有具体的规定和使用方法),然后通过按键盘上的字符键,将源程序按行按字符顺序输入计算机,在输入完一行的字符后要按一次"回车"键(在键盘上一般以"RETURN"或"ENTER"标志),在本书中以"↵"符号代表"回车".

注 本书中符号"'"与符号"'"的作用是相同的.

如果需要从第7列输入字符,则应先按六次"空格"键,使光标移到第7列位置,然后按列依次输入后续的字符.

2. 用卡片输入

大、中型计算机系统采用成批处理程序(简称"批处理")时常用卡片输入. 卡片的形式见图

2.7.

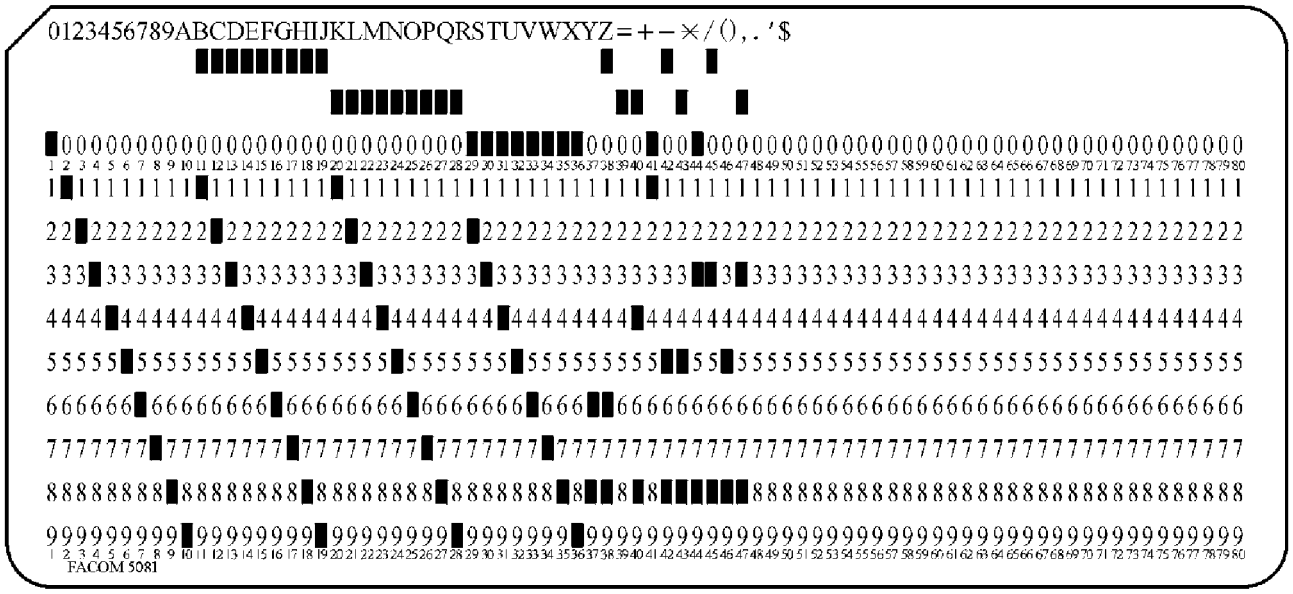


图 2.7

一张卡片有 80 列，可以记载 80 个字符信息. 它相当于源程序纸上的一行. 如果一个源程序有 100 行，则用卡片输入时需要准备 100 张卡片. 在卡片某一列上穿一个到三个孔可以代表该列上某一个字符. 字符与卡片穿孔之间对应关系见附录Ⅶ. 例如“A”的穿孔位置为 12～1，在卡片某一列上的第一行和第十二行处穿孔即可. 在卡片上共有 12 行，其中 10 行用 0 到 9 标志，在 0 行上面为 11 行，11 行上面为 12 行.

用卡片方式，需要加上控制卡片（每种计算机分别规定了控制卡片的格式），然后作为一个作业，放在读卡机上，启动读卡机，卡片上的信息就送入计算机中.

用卡片的好处是便于检查和修改. 如要改一个语句只需重新穿一张卡片代替原有的一张卡片即可.

3. 用软磁盘输入

事先用“键-盘装置”（它是和计算机系统“脱机”的，不相连的）将源程序由键盘输入并记录在软磁盘上，一张软磁盘（8 英寸）可以放 16 万个字符，相当于 2000 张卡片. 将软磁盘放入“软盘读入机”（与计算机联机的），启动软盘机，将盘中信息送入计算机.

用软盘比较经济，一张软盘可以反复使用，携带也方便. 不少大、中型计算机系统采用此方式输入.

也有一些源程序记录在磁带上（特别是一些应用程序是记录在磁带上出售的），需用时将磁带装在磁带机上将信息送入计算机.

§2.7 运行一个 FORTRAN 程序的过程

目前，几乎所有的计算机系统都配有操作系统. 所有程序必须在操作系统支持下才能运行. 在写好了一个 FORTRAN 源程序后，还必须经过编辑、编译、连接、运行等几个步骤. 比运行一个

BASIC 程序要复杂一些.

在每一种计算机系统上实现上述步骤的具体方法不尽相同,但大同小异. 今以长城 0520(与 IBM PC 兼容)为例说明运行一个 FORTRAN 程序的过程.

1. 编辑源程序

所谓编辑,指建立源程序(将源程序建立在磁盘上)和修改源程序. 这个功能是由称为"文本编辑程序"的软件来完成的."文本编辑程序"提供了插入、删除、修改等功能. 在此不一一介绍,用时请查阅说明手册.

长城 0520 或 IBM PC 系统启动后,显示并出现系统提示符"C >"或"A >"(A 和 C 是磁盘驱动器的名称,"C >"表示计算机联接的"当前盘"是 C 驱动器. 当前盘是可以转换的,如果系统显示出"C >",打入"A:"并按回车键,就显示出"A >",表示当前盘已由 C 改为 A),表示系统处于操作系统命令状态,等待输入操作命令. 如果所需的系统软件(包括编辑程序、编译程序、连接程序)已放在硬磁盘上(硬盘驱动器已被定名为"C"),假设源程序想放在用户软盘上,则将该盘放入软盘驱动器 A 中. 先调入"文本编辑程序",在长城 0520 中它是"行编辑程序"(EDLIN),如果我们需要建立的源程序磁盘文件名字叫 A1. FOR(后缀 FOR 表示是 FORTRAN 源程序文件). 则打入以下命令.

A > C:EDLIN A1. FOR ↵(有下划线的部分是用户打入的,下同)其中"C:"表示 C 驱动器中的盘. 上面命令的意思是调入 C 盘上的 EDLIN 行编辑程序,并对当前盘(即 A 盘)上的 A1. FOR 源程序进行编辑.

如果是新建立的文件,则显示屏上会出现:

NEW FILE (表示是新文件)

* (* 是进入编辑状态的提示符)

打入"插入"命令 I(Insert 的缩写. 表示要插入内容)

*I ↵

然后按上一节介绍的办法从键盘上将源程序逐行地键入. 在输入完毕后,按 Ctrl + Break (按住"Ctrl"键再按"Break"键)退出插入状态. 如果需要检查已输入的内容可打入:

*L ↵

列出程序清单. 如发现有错,可通过修改命令进行修改(在此不详述,可参阅有关手册). 如无错则打入 E 命令退出编辑状态,返回操作命令状态,显示屏出现"A >":

*E ↵

A >

2. 编译源程序

长城 0520 对 FORTRAN 源程序要经过两次编译. FORTRAN 编译程序在 C 盘上,而源程序 A1. FOR 在 A 盘上. 为进行第一次编译应键入:

A > C:FOR1 A1;↵

从 C 盘调入 FOR1 编译程序,对 A 盘上的 A1. FOR 源文件进行第一次编译,如有语法错误,

会显示出“有×个错误”的信息. 然后应设法把它们找出来修改(每修改一次都必须重新用 EDLIN 命令,对源程序文件 A1. FOR 进行修改). 直到第一次编译不再出现错误为止. 接着进行第二次编译:

A > C:FOR2 A1;↵

如不发生错误,则全部编译完成. 产生目标程序 A1. OBJ.

3. 连接

编译后得到目标文件. 如果一个程序包含几个程序单位则应分别进行编译,得到几个目的程序. 然后把它们与子程序库连接成为一个统一的目的程序. 如果只有一个程序单位(名为 A1),则可打入:

A > C:LINK A1;↵

如果有几个程序单位(主程序和子程序),名字分别为 A1,A2,A3,则打入:

A > C:LINK A1 + A2 + A3;↵

这是为了使 A1,A2,A3 和函数库联结起来. 显示屏上出现:

Cannot find library A:FORTRAN. LIB

Enter new drive letter:

表示在 A 盘上找不到 FORTRAN 函数库,要求打入函数库所在的盘的驱动器名. 可打入“C↵”. 联结完毕后,产生一个可供执行的目的程序块,系统自动给它定名为 A1. EXE,存放在 A 盘上.

4. 运行

得到名为 A1. EXE 的可执行文件后,就可以直接运行该程序了. 直接打入 A1 即可.

A > A1↵

可得到运行结果.

以上介绍的只是最简单的情况,目的是使读者对运行 FORTRAN 程序的步骤有一个大致的了解. 如果所用的不是 IBM PC/XT(不带硬磁盘而只有 A,B 两个软盘驱动器)的话,以 A 盘为系统盘,B 盘为用户盘,则将上述步骤中的“A”改为“B”,将“C”改为“A”即可. 不同计算机系统的具体规定有些不同,请参阅有关手册.

我们已经建立起了关于 FORTRAN 程序的初步概念,从下一章起将具体介绍怎样利用 FORTRAN语言进行程序设计. 这里包括两方面的内容,一是 FORTRAN 语言的具体规定,二是程序设计的基本方法. 不了解语言的具体规定就不可能正确地写出 FORTRAN 程序,但是学语言的目的是利用它去编写程序. 因此不能满足于“知道语法规则”就算了,而必须善于应用它设计出好的程序来. 在学习方法上不能死记硬背语法规则,而要通过分析例题、编程序练习、上机调试自然而然地掌握主要的、常用的规则,至于一些不常用的、不重要的规则不必硬记,在用时查一下手册即可. 在本书中主要介绍常用的语句(符合结构化原则的语句)和基本的规则,使读者能进行一般的程序设计. 有一些深入的内容(如文件的使用、复杂的输入输出格式)本书不准备作详细介绍,读者在有了一般的程序设计基础以后,需要进一步提高时可参阅有关资料,是不会发生什

么困难的.

只有掌握本课程的特点，并且有正确的学习方法，才能取得好的学习效果.

习　题

1. FORTRAN 程序纸上一行包括哪几个区？每个区内写什么内容？

2. FORTRAN 程序中的标号起什么作用？在什么情况下需要用标号？

3. 试用 FORTRAN 编一个程序求 $\sum_{n=1}^{100} n$ 并写在程序纸上（如果没有程序纸，则应注明字符所在的列数）.

4. 试编程序求 $\frac{1}{1\cdot 2}+\frac{1}{2\cdot 3}+\frac{1}{3\cdot 4}+\cdots+\frac{1}{n(n+1)}$，用程序纸书写.

5. 试编程序求 1 到 100 之间全部奇数之和. 用程序纸书写.

6. 运行一个 FORTRAN 程序需要经过什么步骤？了解你所用的计算机系统的具体规定？如果有可能，在计算机上运行一个 FORTRAN 程序.

第三章　算术表达式和赋值语句

FORTRAN 主要应用于数值计算领域，许多程序的主要功能在于计算，而计算是通过表达式的求值来实现的. 例如下面的 FORTRAN 语句：

$$W=(SIN(X)+COS(X))*L/T+Q*Q*R+0.5$$

它的作用是求出表达式$\frac{(\sin x+\cos x)\cdot L}{T}+Q^2\cdot R+0.5$的值，并赋给变量 W.

因此，表达式是 FORTRAN 程序中不可缺少的成分. 从上面语句可以看到，在表达式中包含常数(0.5)，变量(X,Q,L,T,R)，函数(SIN,COS)和运算符(如 +，*，/). 它们是算术表达式的基本成分.

§3.1　常　　数

所谓常数是在程序执行过程中其值固定不变的量. 在 FORTRAN77 中有以下几种类型的常数：

1. 整型常数　　(如 1，-3,105 等)
2. 实型常数　　(如 1.38，-587.23,1.2E12,它代表 1.2×10^{12})
3. 双精度型常数　　(如 2.8712598765D18 代表 $2.8712598765\times10^{18}$)
4. 复型常数　　(如(3.5,4.2),代表复数 3.5 +4.2i)
5. 逻辑型常数　　(如. TRUE. 代表“真”. FALSE. 代表“假”)
6. 字符型常数　　(如‘BEIJING’,‘WANG’等)

前四种为数值常数，最常用的是整型常数和实型常数，本章只介绍这两种常数. 复型常数和双精度常数将在第十一章中作简单的介绍. 逻辑型常数在第五章中介绍，字符型常数在第八章中介绍.

FORTRAN 常数的表示方式和数学中表示的方法大体相同，但不能以分数形式表示，数字间不得加入逗号，如 1/2，-3/8,1,000,000 不是 FORTRAN 常数.

1. 整型常数(整数,Integer)

整型常数由 0 到 9 的数字组成，它前面允许有一个数符(正或负). 不允许有小数点. 例如 86,3，-6 是整数. 而 10.0，-86.00 不是整数，因为含小数点，FORTRAN 把出现小数点的数按实数处理. 数字间可以出现空格，空格不起作用，如 8765 和 8 7 6 5 等效.

整数在内存中以两个字节(16 位)或四个字节(32 位)存储. 在微型计算机上由于内存量少，一般以二个字节存放一个整数，其中第一个二进位用来代表正负号，后面 31 位用来表示数，因此，一个正数最大值为：

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

即 $2^{15}-1$. 最大的负数为 -2^{15}. 整数的合法范围为 $-2^{15}\sim 2^{15}-1$ 即：$-32768\leqslant N\leqslant 32767$. 一般中、大型计算机系统用四个字节存放一个整数，其允许范围为 $-2^{31}\sim 2^{31}-1$，即，$-2147483648\leqslant N\leqslant 2147483647$，即整数的绝对值应小于 21 亿，有少数计算机（如 DPS-8）字长为 36 位，以 36 位来存储一个整数，因此一个整数的范围为 $-2^{35}\sim 2^{35}-1$，即近似地认为 $|N|\leqslant 3.4\times 10^{10}$.

因此可知，整数的范围是有限的，在编写程序时应估计整数的范围，否则会出现“溢出”的错误.

2. 实型常数（实数，real）

实型常数有两种形式：

（1）小数形式

与日常习惯的形式相同，由数字 0～9 和一个小数点组成，数的前面允许有一个数符（正负号）. 如 17.6，-0.87，+.760，18.，0.006，10.0 等都是合法的 FORTRAN 实数.

（2）指数形式

在数学中，往往用指数形式表示一个大数或小数，例如 3×10^{26}，-7.6×10^{-56} 等. 由于在计算机设备中无法表示上下角字符，故以符号 E 来代表指数. 如以 1.6E2 表示 1.6×10^{2}，以 3E4 表示 3×10^{4}.

可知，以指数形式表示的实数由数字部分和指数部分组成，数字部分可以是小数（如上面的 1.6）或整数（如上面的 3）. 指数部分由一个字母“E”后面跟一个正负号（也可以没有正负号）以及一至二位整数（有的计算机系统可以用到三位整数）组成. 例如：

3.86753	E-12		186	E4
数字部分	指数部分		数字部分	指数部分

数字部分是数的有效数字，指数部分的作用是使数字部分的小数点左移或右移若干位. 数字部分如果是整数，则认为小数点在整数之后. 在小数点左移或右移时，如有必要时在数字之前或之后补以足够的零. 表 3.1 给出了指数形式和小数形式的关系.

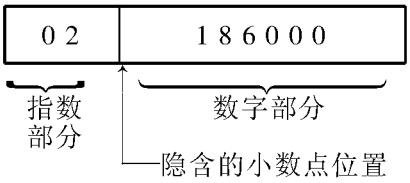
表 **3.1** 指数形式和小数形式的关系

FORTRAN 中指数形式的实数	以科学记数法表示的数	以小数形式表示的数	说　明
1.8E1	1.8×10^{1}	18	指数为 1，将数字部分中小数点右移一位
26.83E-2	26.83×10^{-2}	0.2683	指数为 -2，将数字部分中的小数点左移二位
21.3E4	21.3×10^{4}	213000.0	小数点右移四位，小数点后原有一位数字，故再补三个零
0.62E-2	0.62×10^{-2}	0.0062	小数点左移一位，在数字前面补二个零
0.183E0	0.183×10^{0}	0.183	

指数部分不能单独用来代表一个数，如 E12 不代表 10^{12}（系统会把 E12 当作一个变量名）.

指数只能用整数而不能用小数，如 1.2E1.6 是非法的. 数字部分和指数部分间不必也不能加乘号，1.86 * E13 的用法是错误的.

一个实数可以表示为小数形式，也可以表示为指数形式. 例如，在程序中写 18.6 和 1.86E1 或 0.186E2，1860E－2 等，作用是相同的. 实数（包括小数形式的和指数形式的）在内存中一律以指数形式存放，它的数字部分的值小于 1，小数点后第一位为非零的数字. 例如上面的 18.6 化成 0.186E2，然后存放. 在内存中以四个字节存放一个实数. 不论程序中写 18.6 或 1.86E1，或 1860E－2，均用同一形式存放：



在内存中存放时，小数点的位置固定在数字部分的最左面，上面表示 0.186×10^{2}. 实际上在计算机内是以二进制数形式存放的.

在计算机输出一个数值时，如果指定用指数形式输出，将按“规格化的指数形式”（或称“标准化的指数形式”）输出. 常用的有两种“规格化的指数形式”：

（1）数字部分的值小于 1，而且小数点后的第一位数字不等于 0，例如 0.681E2，－0.99978E－12. 而 21.834E－8，0.0023E4 就不属此.

（2）数字部分有一位（而且只能有一位）非零的整数，即小数点前有一个非零的数字. 例如 6.81E1，－9.9978E－13. 而 0.21834E－6，，123.2E8 则不属此.

不同的计算机系统分别采用上述两种“规格化的指数形式”之一. 在输出时先将实数化成规格化的形式，再打印出来. 转换是由计算机系统自动进行的.

由于实数在内存中是用有限的字节来存放的，因此实数的范围也不可能是无限制的. 一般计算机系统 FORTRAN 允许实数的范围大致为 $|N|\leqslant10^{75}$，有效位为 7～9 位. 也有的只允许 $|N|\leqslant10^{38}$，因此指数最多为二位整数. 有的计算机系统允许数的范围较大，可以超过 10^{100}，因此，指数可以是三位数字. 所以，应该了解所用计算机的规定.

在计算机上可以精确地表示一个整数，不出现误差，而且整数运算的速度快，但整数能表示的范围很小. 用实数形式表示一个数往往会出现误差，例如十进制数 0.1 就不可能用有限位数的二进制数准确地表示. 实数运算速度慢. 但实数能表示的范围远比整数的大. 例如 3×10^{12} 这样的数就无法用整数表示.

因此要根据实际要求来选择用整数还是实数. 如果要求绝对精确而数值不大（如循环次数），则应该用整数，如果数值范围大则用实数形式. 请注意，123 和 123.0 在数学上是等价的，但在 FORTRAN 中它们的作用是不同的，前者以整数形式存放，后者以实数形式（指数形式）存放.

§3.2 变　　量

在程序执行过程中其值可以变化的量称为变量. 一个变量用一个变量名作标志. 一个变量在

内存中以一个存储单元存放其值. 如：

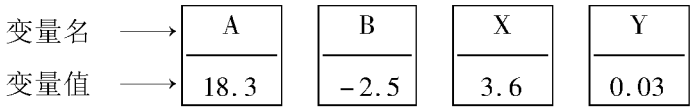


图 3.1

在程序运行过程中，可以用一个新值代替一个变量的原值. 如有下面二个语句.

A = 18.3

A = 20.6

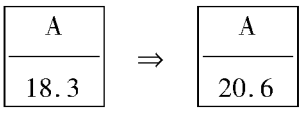


图 3.2

在执行第一个语句后 A 的值（或者说“A 的内容”，即存放在变量 A 中的数据）为 18.3，在执行完第二个语句后 A 的值变为 20.6. 一个变量在某一瞬时只能有一个固定的值. 在程序中用到的变量一定要给它一个固定的值，否则计算机会使它有一个不确定的值（而不是零或空白）.

FORTRAN 的变量名必须以字母（A 到 Z 中的字母）开头，后跟 5 个以内的字母或数字，例如 A，B，C3，G86，MASS，WEIGHT，RATE，AMOUNT，G34T 等都是合法的变量名，而 3A，3G8，X + Y，LI - LA，1986.3 就不是合法的 FORTRAN 变量名. 如果变量名长度超过 6 个字符，有的计算机系统按出错处理，有的则只取其前 6 个字符.

为了改善程序的可读性，便于理解变量的作用，建议尽量用“见名知义”的变量名. 如以 AMOUNT代表“金额”（存放“金额”的值），以 RATE 代表利率，以 WEIGHT 代表重量，以ALFA代表 α，以 PI 代表 π 等.

FORTRAN 常数有六种不同类型，变量也相应地有六种类型，以便存放相同类型的数据. 例如用整型变量存放整型常数，用实型变量存放实型常数. 整型变量所占内存的字节数和存放数据的形式与整型常数相同. 同样，实型变量占的字节数和存放数据的形式与实型常数相同. 不能用整型变量存放一个实数，反之亦然.

在程序中应当给每一个变量确定一个类型. 确定变量类型的方法有以下三种：

（1）隐含约定

FORTRAN 规定，凡不另加专门说明时，系统将按变量名的第一个字母来确定变量的类型. 凡名字以 I，J，K，L，M，N 这六个字母开头的变量均自动作为整型变量，以其他字母开头的变量作为实型变量. 例如 I，IMAX，MAN，MI，KING，N8 等均为整型变量，而 TIME，AVER，AMOUNT，VALUE，G3，X，Y，Z，A 等均为实型变量. 可以将这个“隐含约定”称为 I - N 规则（以 I 到 N 之间的字母开头的变量为整型）.

（2）用 IMPLICIT 语句（隐含说明语句）确定变量类型

可以用 IMPLICIT 语句改变以上隐含约定确定的变量类型. 如想将 X 到 Z 开头的变量也作

为整型变量，可以用

```
IMPLICIT INTEGER (X,Y,Z)
```

或

```
IMPLICIT INTEGER (X-Z)
```

如果想使I,K,N开头的变量作为实型变量，可用

```
IMPLICIT REAL (I,K,N)
```

可以用一个IMPLICIT语句同时确定几种类型，如：

```
IMPLICIT INTEGER (X,Y,A),REAL (L,K),INTEGER (A-D)
```

（3）用类型说明语句确定单个变量的类型

有时只需改变某一变量的类型.例如以IMAX代表“最大电流值”，按I-N规则IMAX为整型变量，但实际电流值不一定恰好是整数.当然可以不用IMAX名而改用AMAX，以使其为实型变量，但这又破坏了程序的可读性，使人难以理解AMAX是代表什么含义.为了保留IMAX名字又使其为实型，可以专门指定它为实型变量.改变单个变量类型的语句称为“类型说明语句”，它包括六个语句：INTEGER语句（整型变量说明语句）、REAL语句（实型变量说明语句）、DOUBLE PRECISION语句（双精度变量说明语句）、COMPLEX语句（复型变量说明语句）、LOGICAL语句（逻辑变量说明语句）、CHARACTER语句（字符变量说明语句）.在这里只先介绍INTEGER语句和REAL语句.例如有以下两个语句：

```
INTEGER A,X1,TIME,M
REAL IMAX K2,AMOUNT
```

规定变量A,X1,TIME,M为整型变量，IMAX,K2,AMOUNT为实型变量.M和AMOUNT本来无须再加说明（M本来就是整型变量，AMOUNT本来就是实型变量），但有时为了读程序时一目了然，将程序中全部变量都明确说明一次.

说明：

（1）如果以上三种方法不一致时，以类型说明语句为最优先，IMPLICIT语句次之.例如：

```
IMPLICIT INTEGER (A,B)
REAL A1
```

所有以A和B字母开头的变量为整型，而A1为实型.

（2）类型说明语句和IMPLICIT语句都是非执行语句，它们的作用仅是通知编译系统将变量按指定的类型分配存储单元和确定数据的存放方式，此外并不产生具体的操作.

（3）类型说明语句和IMPLICIT语句应出现在程序中可执行语句的前面，而IMPLICIT语句又应在类型说明语句之前（见附录Ⅳ）.

（4）类型说明语句和IMPLICIT语句只在本程序单位内有效.假如一个主程序中有一变量X，子程序中也有一个变量X，这两个X虽然同名，但不代表同一变量.如果在主程序中用INTEGER X语句说明X为整型变量，那么它不影响子程序中变量X的类型，子程序中的X仍为实型.

§3.3 算术运算符

1. 算术运算符

FORTRAN 有五种算术运算符，如表 3.2 所示.

表 **3.2**

运 算 符	意 义
+	加（或正）
−	减（或负号）
*	乘
/	除
**	乘方

请注意，乘号是用“ * ”而不是“ × ”，以免与字母 X 混淆. 乘号是不能省略的，例如不能以 ABC 代表 A * B * C. 除号必须用“/”符号，用两个“ * ”号代表乘方运算，2^3 以 2 * * 3 表示. 两个运算符不能相邻，例如$\frac{A}{-B}$不能写成 $A/-B$，而应写成 $A/(-B)$.

2. 算术运算符的优先顺序

优先顺序如表 3.3 所示.

表 **3.3**

运 算 符	优先级别
**	最优先
* 和/	次之
+ 和 −	最低

同一优先级的按从左到右原则. 例如：

```
2.5 +3 * A /T +V **2
    ↑  ↑  ↑  ↑   ↑
    ④  ②  ③  ⑤   ①
```

先进行乘方运算，再进行乘法，第三步是除法运算，最后是加法运算（先左后右）.

−A * * 2 相等于 −(A * * 2). A/B * C 相等于(A/B) * C. 对 3 * * 2 * * 3，是先计算 3^2 还是先计算 2^3 呢？FORTRAN 规定，对多次乘方，先右后左，即先进行 2^3 运算，得结果 8，然后再进行 3^8 运算，得 6561. 它相当于 3 * *(2 * * 3).

§3.4 内部函数简介

在解题中常会遇到一些专门的运算，如求 sin 和 cos、平方根、对数、绝对值，求几个数中最大数和最小数，取一个数的整数部分等. 有的还要进行多次. 在 FORTRAN 中提供了一系列的函数来完成各种功能的运算. 程序设计者不必自己编写程序段来完成这些操作，只需写出一个相应的函数名字和代入所需的参数即可. 例如只需写出 SQRT(4.0)，就会计算出$\sqrt{4}$的值 2，写出 SIN(0) 就能得到 0 的正弦值 0，FORTRAN 提供的函数见本书附录Ⅱ. FORTRAN 将这些函数分别编成一个个子程序，组成函数库，作为编译程序的一部分，用 LINK 命令实现各程序单位与函数库的联

接. 在 FORTRAN 中，系统所提供的函数称为“内部函数”，又称“库函数”.

FORTRAN 内部函数一览表请见附录Ⅱ. 在此只介绍几种最常用的函数(见表 3.4).

表 **3.4** 常用内部函数

函 数 名	含　义	数学符号	FORTRAN 例
ABS	求绝对值	$\|a\|$	ABS(A)
EXP	求指数函数的值	e^x	EXP(X)
SQRT	平方根	$\sqrt{a}$	SQRT(A)
SIN	正弦值	$\sin x$	SIN(X)
ASIN	反正弦	$\sin^{-1}a$	ASIN(A)
COS	余弦值	$\cos x$	COS(X)
ACOS	反余弦	$\cos^{-1}a$	ACOS(A)
TAN	正切	$\tan x$	TAN(X)
ATAN	反正切	$\tan^{-1}y$	ATAN(Y)
LOG	自然对数	$\log_e(a)$	LOG(A)
LOG10	常用对数	$\log_{10}(a)$	LOG10(A)
INT	取整	$\text{int}(a)$	INT(A)
MOD	求余	$a_1-\text{int}(a_1/a_2)*a_2$	MOD(A1,A2)
SIGN	符号	$\begin{cases}\|a_1\| & (\text{若 } a_2>0)\\ -\|a_1\| & (\text{若 } a_2<0)\end{cases}$	SIGN(A1,A2)
REAL	转换为实型		REAL(I)
MAX	取最大值	$\max(a_1,a_2,a_3)$	MAX(A1,A2,A3)
MIN	取最小值	$\min(a_1,a_2,a_3)$	MIN(A1,A2,A3)

例如，有关函数的值如下：

ABS(−5.6)=5.6

EXP(3.0)=e^3=20.08553

SQRT(4.0)=2.0

SIN(1.0)=sin(57.29578°)=0.8414709

COS(1.0)=cos57.29578° =0.5403023

TAN(1.0)=tan57.29578°=1.557408

ATAN(1.0)=0.7853981(弧度)

LOG(3.0)=1.098612

LOG10(100.0)=2.0

INT(3.6)=3

MOD(8,6)=2

SIGN(3.0,−2.0)=−3.0

SIGN(−3.0,2.0)=3.0

REAL(6)=6.0

MAX(3,6,1,10,5) = 10

MIN(3,6,1,10,5) = 1

说明：

（1）每一个内部函数要求一个或几个自变量，自变量应用括弧括起来. 大部分函数只要求一个自变量（如 SIN,COS,LOG 等），有的函数要求二个自变量（如 MOD,SIGN），有的函数则要求不少于二个的自变量（如 MAX,MIN）. 在附录 II 中可以查出各个内部函数要求自变量的个数. 其实，根据函数的作用就可以判断出自变量的个数，而不必死记，甚至不必查表即可知. 例如，求 SIN,COS,TAN,LOG,EXP 等函数的值显然只允许一个自变量，而 MOD 函数是“求余”函数，MOD（A1,A2）的作用是求 A1 除以 A2 的余数（MOD(8,6)的值为 2），显然需要二个自变量. MAX 和 MIN 是从若干个数值量中取其中值最大者和最小者，显然它要求二个或多个自变量. 如果函数要求有一个以上的自变量时，请注意自变量的顺序对运算结果有无影响. 如 MOD(A1,A2)，A1 是被除数，A2 是除数，A1 和 A2 的顺序不能颠倒. 而 MAX(X1,X2,X3)，得到 X1,X2,X3 中的最大值，与自变量的顺序无关.

（2）函数对自变量的类型是有要求的. 例如 SIN,COS,TAN,LOG,LOG10,EXP,SQRT 等要求自变量为实型（也可以是双精度或复型的量）但不能是整型. 因此 SIN(1.0)，SQRT(4.0)，LOG(3.0)是合法的，而 SIN(1)，SQRT(4)，LOG(3)是不正确的用法.

（3）FORTRAN77 函数的值也是有类型的. 例如 SQRT(4.0)的值是实数 2.0 而不是整数 2. 函数值的类型是与自变量的类型有关的. 例如 SIN,COS,SQRT,LOG,EXP,MAX,MIN 等函数，如果自变量是实型的，则得到的函数值也是实型的，如果自变量是双精度型或复型的，则得到的函数值也是双精度型或复型的. ABS(-3.0)的值为实数 3.0，而 ABS(-3)的值为整数 3. 使用灵活方便. 详见附录 II. 如记不住，一般可用实型自变量.

（4）三角函数的角度单位是“弧度”而不是“度”. 例如，SIN(6.0)不代表 sin6°，而代表 6 个弧度的正弦值. 同样，ATAN 得到的角度值也不是用度表示而用弧度表示，因此，ATAN(1.0)的值不是 45°而是用 0.7853981 弧度表示.

（5）INT 的作用是取一个实数的整数部分，即将小数点后的数截去. INT(3.5)得 3，INT(-3.5)得 -3，而不是 -4，这是与 BASIC 中的 INT 函数不同的. REAL 函数的作用是将一个整型数转换成实型数，REAL(6)得到 6.0. 它们在内存中存放的形式不同.

（6）SIGN 函数的作用是“符号传送”. SIGN(A1,A2)的作用是将 A2 的符号传送给 A1. 例如 SIGN(3.0,-2.0)，将后一项的符号（负号）传给第一项，得值 -3.0. 而 SIGN(-3.0,2.0)则把后一项的正号传给第一项，得值 +3.0. 如果第二项的值为零，则不发生符号传送.

（7）自变量可以是常数、变量或表达式. 例如，SQRT(16.0)、SQRT(T)、SQRT(16.0+4.0+T/2.0)都是合法的. 甚至可以嵌套使用，如 SQRT(SQRT(16.0))，其值为 2.0.

（8）FORTRAN77 中函数的使用方法是比较简单的（见上述）. 由于考虑到与 FORTRAN66 的兼容，保留了 FORTRAN66 中的函数名. 因此函数名分为“属名”和“专用名”两种（详见附录 II）. 前面我们介绍的都是“属名”. 有的 FORTRAN 子集只能使用“专用名”. 请注意所用系统的规定.

§3.5 算术表达式

1. 表达式是由运算符和括号将各运算元素(也称操作数.例如,常数、变量、函数、数组元素等都是基本的运算元素)连结起来的有意义的式子.FORTRAN 有四种表达式:算术表达式;关系表达式;逻辑表达式;字符表达式.算术表达式中各运算元素都必须是算术量,用算术运算符和括号把它们连接起来.如果 TIME 和 C 都是数值变量,则:

3.2 + 8.5 * TIME * * 2/(- C)

就是一个算术表达式.算术表达式的值是一个数值.

最简单的算术表达式只有一项,例如只包括一个常数或一个变量.如:

A,8.2,SIN(X),TIME

都是表达式的一种形式.

表 3.5 是一些数学表达式表示成 FORTRAN 算术表达式时正确和错误的写法.

表 **3.5**

数学表达式	FORTRAN 算术表达式	
	错　　误	正　　确
$\frac{a+b}{c+d}$	A + B/C + D	(A + B)/(C + D)
$\frac{u-v}{-c^2}$	U - V/ - C * * 2	(U - V)/(- C * * 2)
$\cos(3x)^2$	COS(3 * X) * * 2	COS((3 * X) * * 2)
$\sin 4t$	SIN4 * T	SIN(4 * T)
$\frac{a \cdot b}{c \cdot d}$	A * B/C * D	A * B/(C * D)
$\sqrt{b^2-4ac}$	SQRTB * * 2 - 4A * C	SQRT(B * B - 4 * A * C)
$a \cdot e^x$	A * E * * X	A * EXP(X)
$12.6\log 37$	12.6 * LOG(37)	12.6 * LOG10(37.0)

注意:

(1) FORTRAN 中只有一种括弧符号“(”和“)”,而没有大、中、小括号之分.数学上的[(a + b)·(c + d)] ×2 在程序中应写成((A + B) * (C + D)) * 2.

(2) 表达式的求值运算的优先次序是:

括号 → 函数 → * * → *,/ → +,-

例如,COS(X + Y) * * 2 的运算次序是,先进行括弧内的运算,然后进行余弦的运算,计算出 cos(x + y)的值,最后才进行乘方的计算.求出 $\cos^2(x+y)$ 之值.如果要求的是 $\cos(x+y)^2$,应该怎样写 FORTRAN 表达式?请读者自己完成它.

2. 在表达式运算中必须注意类型的问题.能不能在一个 FORTRAN 表达式中进行不同类型量之间混合运算呢?在 FORTRAN66 标准中是不允许的,而 FORTRAN77 是允许的.例如:

I * J/T + 3.6/2

I,J,2 为整型,T,3.6 为实型. 这是允许的. 那么按什么规则进行运算呢?

(1) 如果运算量类型相同,则运算后仍保持原类型. 如 5 ×6 结果为 30,仍为整型,3.6 + 7.4 结果为 11.0,为实型. 2 * * 3 结果为 8,为整型.

值得注意的是:两个整型量相除结果仍为整型量. 如,5/2,结果不是 2.5,而是整数 2,这是因为只取商的整数部分而将小数部分舍去,-6/4 的结果是 -1. 这和数学上是不同的,如不注意就会出错. 例如:$16^{\frac{1}{2}}$不要写成 16 * *(1/2). 因为 1/2 结果为 0(而不是 0.5),因此 16 * *(1/2)的结果不是 4 而是 16^0,等于 1 了. 同理,$\frac{1}{2}vt^2$ 不要写成(1/2) * V * T * * 2,结果也成零了. 想要避免这种情况,应改用实型量相除. 如,5.0/2.0 结果得 2.5,-6.0/4.0 结果为 -1.5,1.0/2.0 结果为 0.5. 所以 16 * *(1.0/2.0)结果就不会是 1 而是 4.0.

在进行整型量的乘除时,应十分注意其顺序. 如数学上$\frac{6\times10}{5}$,既可写成$\frac{6}{5}\times10$,也可写为 6 ×10 ÷5,结果相同. 但在 FORTRAN 中 6/5 * 10 的结果是 10,而 6 * 10/5 的结果是 12. 二者结果不同. 不要随便地在整型运算中使用数学上的交换律. 在实型量的运算中就不会出现这种情况,如,6.0/5.0 * 10.0 和 6.0 * 10.0/5.0 结果是相同的.

因此,在编写程序时,不要轻易将两个整型量相除.

一个整型量的整数次方,结果也是一个整数,如,3 * * 2 结果为 9,3 * *(-1)的结果不是 0.3333333 而是 0,8 * *(-2)的结果也是零. 这是由于在运算后将小数部分舍去了.

如果一个算术表达式中所有运算量都是整型量,则称为整型表达式,其结果为整型. 同样,如果表达式中所有的运算量为实型,则称为实型表达式,结果为实型.

(2) 不同类型的数值量混合运算时,编译系统会先将它们转换为同一类型的量然后进行运算. 转换的原则是:将低级的类型转换为高级的类型. 类型的级别如下:

整型 ——→ 实型
(低)　　(高)

(双精度型和复型的级别比实型高)

如果一个整型数与一个实型量相加,则先把整型数化为实型数,然后相加. 例如,3 + 9.6,先将 3 化成 3.0,然后将 3.0 和 9.6 实行同类型运算. 加减乘除的混合运算规律见表 3.6.

例如,I * J,I - 5,K/8,8 + 3 的结果均为整型,而 8 * Y,13 + Q,6/2.0,T/R 的值为实型.

乘方运算的规律见表 3.7.

例如,2^2 的值为整数 4,$2^{2.0}$的结果为实数 4.0,2.0^2 的结果为 4.0,$2.0^{2.0}$的结果为 4.0. 以上

表 **3.6**

运算结果 / 运算量 1 / 运算量 2	整　型	实　型

整　型	整　型	实　型
实　型	实　型	实　型

表 **3.7**

乘方运算结果：基数类型 \ 指数类型	整　型	实　型
整　型	整　型	实　型
实　型	实　型	实　型

规律也不必死记，因为一般情况下整型数的实数次方（如 $2^{2.5}$）显然是实数，同样实型数的整数次方（如 2.5^2）一般也是实数.

在进行乘方运算时，如果指数为整数按自乘处理，如，3 ＊＊2 是按 3 ＊3 处理的，3.0 ＊＊3 是按 3.0 ＊3.0 ＊3.0 处理的，这种不属于混合运算. 如果指数为实数，则按下面方法计算：

$$A^B = e^{BlnA}$$

即先对底数取对数得 lnA，再乘以 B，再计算出 e^{BlnA} 之值，它就是所求的 A^B 的值. 由于要计算对数和指数，运算速度慢. 如果指数为整型的，则运算速度快. 因此，如果指数为整数的话不要写成实数形式. 例如，2.5^2 应写为 2.5 ＊＊2 而不要写成 2.5 ＊＊2.0.

类型的转换是按表达式的运算顺序进行的. 如：

$$1/2 * 3.6 - 1.2$$

因为从左而右地运算，先进行 1/2，得零，因此表达式结果为 －1.2. 如果表达式改写为：

$$3.6 * 1/2 - 1.2$$

则结果为 0.6. 因为先进行 3.6 ＊1 的运算，先将 1 化成 1.0，3.6/1.0 得 3.6，再进行除法，也先将 2 化成 2.0，得 1.8，然后做减法.

混合运算的例子见表 3.8.

表 **3.8**

表　达　式	值	说　　明
6/4 ＊3.0 ＋2	5.0	6/4 值为 1，整-实运算，结果实型
3 ＊2 ＊＊3/12/5.0	0.4	24/12 结果为 2，除以 5.0，结果实型
3.6 ＊1/2 ＊＊2	0.9	3.6/4 结果实型
3.6 ＊（1/2 ＊＊2）	0	1/4 值为 0
1/3 ＋1/3 ＋1/3	0	1/3 的值为 0
1.0/3 ＋1.0/3 ＋1.0/3	0.9999999	1.0/3 结果为实数 0.3333333

由于混合运算转换数据类型需要花时间，因此混合运算比相同类型的运算速度慢. 总之，整型量的运算最快，实型之间的运算次之，混合运算最慢. 例如，4.0 ＊6.5 比 4 ＊6.5 运算快一些. 如果一个表达式在程序运行过程中被多次执行的话，最好将它写成同一类型. 但是在执行次数不

多的情况下，为了使程序具有易读性，往往采用混合运算的形式.

3. 在表达式的计算中要注意由于有效位数的限制而引起的误差.

用计算机进行数值计算，有时是会产生一些微小的误差的，这是由于计算机用以存放数据的存储单元是由有限的字节（或二进位）组成的. 因此不可能容纳无限多位数的数，也就是说，其有效位数是有限的（一般为十进制数的 7 ~ 9 位）. 例如 1.0/3.0 的结果在内存中不是 0.333333333333……，而是 0.3333333（假定所用的计算机系统的有效位数为 7 位），与理论值有一些小的误差. 这是在程序设计中事先应估计到的.

如果有以下的计算：

$$1234567. + 0.001 - 1234561.$$

理论值应为 6.001. 但由于所用的计算机系统有效位数为 7 位，不能存放 1234567.001（前两项之和）这个有效位数为 10 位的数，而把多余的位数截去而保留 1234567，再减去 1234561. 得 6.0. 如果把运算次序改变一下：

$$1234567. - 1234561. + 0.001$$

就可得到 6.001. 因为前二次之差为 6.0，有效位数不超过 7 位，全部保留. 可见，不要使两个相差很大的数直接相加或相减.

还应当考虑到表达式计算过程中会不会产生数的“溢出”，所谓“溢出”指的是超过了存储单元所允许的数的范围. 例如：

$$1.2E45 * 1.6E43/2.0E20$$

前二项的乘积是 1.92E88，已超过了一般计算机系统所允许的实数范围了，故要发生“溢出”. 有的计算机系统此时使程序停止运行按出错处理，有的则将溢出的数按该计算机系统可表示的最大实数处理（如 1×10^{75}），但它已无意义了，因为它产生了潜在的错误而难以发现. 如果改变运算次序，先进行除法：

$$1.2E45/2.0E20 * 1.6E43$$

前二项之和为 0.6×10^{25}，未溢出，再乘上第三项得 0.96×10^{68}. 如果所用的计算机系统实数表示范围可达 1×10^{75} 的话，它仍未溢出，可得正确计算结果.

因此写表达式时，要避免两个很大的数相乘，或一个很大的数除以一个很小的数.

§3.6 赋值语句

赋值语句的作用是将一个表达式的值赋给一个变量（或一个数组元素）. FORTRAN77 的赋值语句有三种：算术赋值语句、逻辑赋值语句、字符赋值语句. 例如：

```
A = 13.6        （算术赋值语句，A 为实型变量）
G = .TRUE.      （逻辑赋值语句，G 为逻辑变量）
C = 'ABC'       （字符赋值语句，C 为字符变量）
```

在本节中只介绍算术赋值语句，其他两种赋值语句将在第五章和第八章中介绍.

算术赋值语句的一般形式为：

$$V = e$$

其中 V 为变量名(或数组元素名),e 为算术表达式. 例如:

```
F = M * A
S = V0 * T + 0.5 * A * T * * 2
L = 2 * PI * R
```

都是算术赋值语句.

算术赋值语句是程序中实现计算的主要语句. 几乎每一个程序中赋值语句都是不可少的. 必须熟练地掌握它的应用.

说明:

1. 赋值语句中的“ = ”号,是“赋值号”,它的作用是将“ = ”号右边表达式的值赋给“ = ”号左边的变量. 它不同于数学中的“等于号”. 以下两个语句作用是不同的:

```
A = B
B = A
```

第一个语句的作用是将 B 的值赋给变量 A. 因此赋值后 A 和 B 的值都是 4. 第二个语句的作用是将 A 的值赋给变量 B,赋值后 A 和 B 都是 3.(如图 3.3)

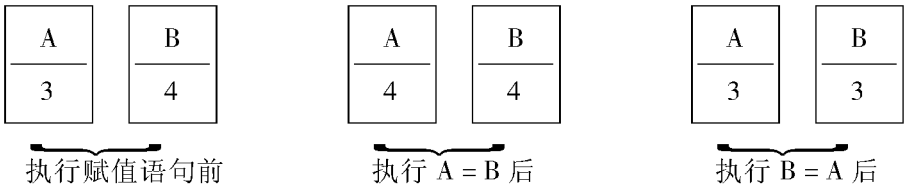


图 3.3

赋值号的左边只能是变量(或数组元素)名,不能是表达式. 例如:

```
A + B = C + D
```

是错误的.

下面语句的作用是将 N 的原值加 1 再送回 N 中去:

```
N = N + 1
```

如果 N 的原值为 5,将它加 1 再送回 N 中去. 6 取代了 N 的原值 5(如图 3.4 所示).

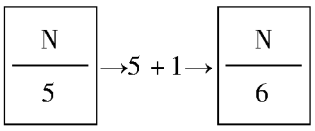


图 3.4

2. 算术赋值语句的执行顺序是:先计算赋值号右边表达式的值,然后将它赋给赋值号左边的变量. 也就是说,赋值语句具有计算和赋值双重功能,先计算,后赋值.

3. 算术赋值语句的类型转换问题

赋值语句中“赋值号”右侧表达式的类型和赋值号左侧变量的类型可以相同也可以不相同.

(1)如果类型相同,则直接进行赋值. 如:

```
N = 100      (N 是整型变量)
T = 2.5 * 8  (T 是实型变量)
```

(2)如果类型不同,则先将表达式的值转换成被赋值的变量的类型,然后进行赋值. 例如:

```
IY = 1.2 * 2.5/2
```

表达式是混合型表达式，其值为 1.5，而 IY 为整型，因此先将 1.5 转换成整型，得值 1，将 1 赋给 IY.

由此可见，赋值语句的执行次序是：先计算，后转换，最后赋值. 例如：

```
A = 2 * 6/5
```

表达式的值为整数 2，而 A 为实型变量，先将 2 化成 2.0 再向 A 赋值.

赋值语句的赋值规则见表 3.9.

表 **3.9**

执行规律：表达式类型 / 变量类型	整　型	实　型
整　型	照赋，结果为整型	先取整，再赋值，结果为整型
实　型	先化实，结果为实型	照赋，结果为实型

请注意，当赋值号两侧的类型不同时，赋给变量的值可能会与程序设计者所想像的不同，如：

```
I = 3.6 * 5 + 1.5
```

I 的值不是 18.5 而是 18. 这往往是引起运行结果错误的一个原因. 因此最好保持两侧类型一致. 例如可以将 I 改名为 AI，或者用 REAL 语句指定 I 为实型变量.

§3.7　STOP 语句、END 语句和 PAUSE 语句

1. STOP 语句（停语句）

它的作用是“停止运行”，是一个执行语句. 一个程序中可以有任意个 STOP 语句（在 FORTRAN77 程序中可以不包含 STOP 语句，而用 END 语句使程序停止运行）

它的形式为：

```
STOP[n]
```

其中 n 是不超过 5 位的数字或一个字符串. 如：STOP 10，STOP 25，STOP‘ABC’均为合法. 在执行到此语句时，除停止程序运行外，还显示（打印）出 n 的值，如上述语句会显示出“10”，“25”或“ABC”等字符，以便使程序员辨别程序的流程.

2. END 语句（结束语句）

在 FORTRAN 66 中 END 不是执行语句，而只作为一个程序单位的结束标志. 在 FORTRAN 77中，END 是执行语句，它既具有 STOP 语句的作用（停止程序的运行），又标志本程序单位的结束. 一个程序单位只能有一个 END 语句（在子程序中 END 语句不兼有 STOP 语句的功能，而兼有 RETURN 语句的功能）.

3. PAUSE 语句（暂停语句）

它的作用是“暂停执行程序”，而不是“停止程序执行”，系统只是把此程序暂时“挂起来”，等待程序员打入命令. 可以使之从断点恢复执行状态，继续执行程序. 各个计算机系统恢复运行

状态的方法和命令是不同的，可查阅说明书. PAUSE 语句的一般形式为：

PAUSE［n］

其中 n 的含义与 STOP 语句中的 n 相同.

在调试程序时，可以事先有意设几个“断点”，分段调试，调试结束后再删去 PAUSE 语句.

§3.8 程序举例

例 1 有一直流电路，电压 U = 100 伏，R_1 = 20 欧，R_2 = 50 欧，R_0 = 100 欧，求电路的等效电阻 R 和总电流 I（见图 3.5）.

从物理学可知，并联电阻（R_{12}）的值为：

$$R_{12}=\frac{R_1R_2}{R_1+R_2}$$

总电阻 $R=R_0+R_{12}$

电流 $I=\frac{U}{R}$

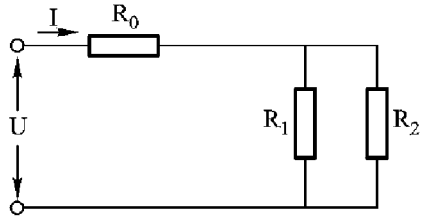


图 3.5

程序可按此编写如下：

```
PROGRAM EXAM1
REAL I
R0 = 100.0
R1 = 20.0
R2 = 50.0
U = 100.0
R12 = R1 * R2/(R1 + R2)
R = R0 + R12
I = U/R
WRITE( * , * )'R = ',R
WRITE( * , * )'I = ',I
END
```

运行结果为：

```
R =          114.2857000
I =    8.750000E - 001
```

以上是在 IBM PC 机上打印的结果，I 的值为 0.875，以指数形式表示. 在不同的计算机系统上，实数表示的方式可能有所不同.

例 2 有一多边形土地，其边长为 l_1 = 100 米，l_2 = 130 米，l_3 = 100 米，l_4 = 140 米，l_5 = 140 米，对角联线长 l_6 = 210 米，l_7 = 205 米. 求其面积（见图 3.6）.

五边形可以分为三个三角形，每个三角形的面积为：

$$A=\sqrt{s(s-a)(s-b)(s-c)}$$

其中 $s=\frac{a+b+c}{2}$. a,b,c 为三角形三个边长.

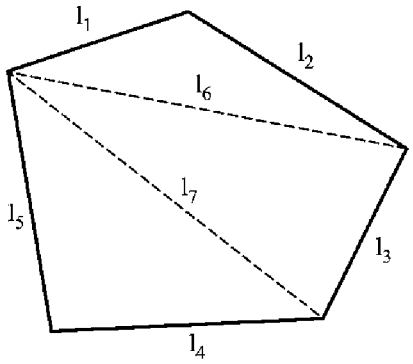


图 3.6

程序如下：

```
PROGRAM EXAM2
IMPLICIT REAL(L)
L1 = 100.0
L2 = 130.0
L3 = 100.0
L4 = 140.0
L5 = 140.0
L6 = 210.0
L7 = 205.0
S1 = (L1 + L2 + L6)/2.0
S2 = (L3 + L6 + L7)/2.0
S3 = (L4 + L5 + L7)/2.0
A1 = SQRT(S1 * (S1 - L1) * (S1 - L2) * (S1 - L6))
A2 = SQRT(S2 * (S2 - L3) * (S2 - L6) * (S2 - L7))
A3 = SQRT(S3 * (S3 - L4) * (S3 - L5) * (S3 - L7))
AREA = A1 + A2 + A3
WRITE(*,*)' AREA =',AREA
END
```

程序选 L1,L2,…,L7 为变量名，以便与题目给的 $l_1,l_2,\cdots l_7$ 相一致，增加可读性. 但事实上长度不一定恰好是整数，故用 IMPLICIT 语句将名字以 *L* 打头的变量指定为实型，S1,S2,S3 代表三个三角形的$\frac{a+b+c}{2}$. A1,A2,A3 为三个三角形的面积，AREA 为总面积.

运行结果为：

```
AREA =     24705.6400000
```

例 **3**　人造卫星绕地球表面做圆周运动，卫星轨道距地面 h 米高时圆周运动速度 v_c 为：

$$v_c=\frac{7900\sqrt{R}}{\sqrt{R+h}}\quad(\text{米/秒})$$

R 是地球半径约等于 6.37154×10^6 米.

在某一特定高度上，卫星脱离轨道的速度为：

$$v_e=v_c\sqrt{2}$$

今要求当 h = 20000 米时，计算卫星的圆周运动速度和脱离轨道的速度(用公里/小时)和环行一周所需时间(用分表示). 以上所得之数据均取至小数点后二位数字，对第三位小数四舍五入.

程序如下：

```
PROGRAM EXAM3
H = 20000.0
R = 6.37154E6
```

```
VC = 7900.0 * SQRT(R)/SQRT(H + R)(圆周速度 V_c,单位为米/秒)
VE = VC * SQRT(2.0)(脱离轨道的速度 V_e,单位为米/秒)
C = 2.0 * 3.1415926 *(H + R)(飞行一周的距离)
T = C/VC(飞行一周的时间,单位为秒)
VC = INT(VC * 3600/1000.0 * 100.0 + 0.5)/100.0(化为公里/小时,并取二位小数)
VE = INT(VE * 3600/1000.0 * 100.0 + 0.5)/100.0(同上)
T = INT(T/60.0 * 100.0 + 0.5)/100.0(化成分,并取二位小数)
WRITE(*,*)'VC = ',VC,'KM/H'
WRITE(*,*)'VE = ',VE,'KM/H'
WRITE(*,*)'TIME = ',T,'MINUTES'
END
```

程序中第八、九行将 v_c 和 v_e 化成“公里/小时”单位应乘以常数$\frac{3600}{1000}$,第十行将 T 由“秒”单位化成“分”单位应除以 60. 这三行都用同一方法把数值取到小数点后二位,并对第三位小数四舍五入. 此方法如下:设想对 A 进行以上的四舍五入处理,可用以下式子:

$$\text{INT}(A * 100.0 + 0.5)/100.0$$

例如,A = 3.1415926,先乘以 100,得 314.15926,再加 0.5 得 314.65926,再取整,得 314,再除以 100,得 3.14,达到要求. 对其他数,请读者自己验证一下. 请注意,用以上方法只能对正数进行四舍五入. 对负数的四舍五入(如 -3.678 希望变成 -3.68)应如何处理,请读者思考.

程序运行结果为:

```
VC   =    28395.4700000 KM/H
VE   =    40157.2600000 KM/H
TIME =       84.8600000 MINUTES
```

习　　题

1. 以下哪些是符合 FORTRAN 规定的整数,哪些是实数?哪些不合法?

-6, 8.747, 8.0, 2/3, 1E8, 0.3E-6, -123, SQRT(4), E-3, 10., 0.034, .34, (1+4), 10**2

2. 以下哪些是符合 FORTRAN 规定的变量名?哪些属整型变量?哪些属实型变量?哪些不合法?

SHANGHAI, LI-LI, 3G4, APPLE, B707, IBM PC, 1986.5, WANG, IMAX, MEN, TRS, AX

3. 将以下数学表达式用 FORTRAN 表达式表示:

(1) $(t-32)\frac{5}{9}$

(2) $\sqrt[3]{5}\left(\cos\frac{4\pi}{3}\right)$

(3) $\cos^3\varphi - 3\cos\varphi\sin^2\varphi$

(4) $\frac{2y^{a+3}}{a-b} + \frac{3x^{4-c}}{c-d}$

(5) $e^{12.6} \cdot \log 3 - 8.6$

(6) $e^{\alpha x}(a\cos\beta x + b\sin\beta x)$

（7）$\frac{6.5^2x^4}{(x-y)\cdot t}\times\frac{3\ln 3}{t-3}$

（8）$(-a^{23}+\sqrt{b})\cdot(a-b)^{-\frac{1}{2}}$

4. 假定各变量类型遵循 I－N 规则，且 I＝3，J＝6，A＝8.5，X＝0.5，请写出下列表达式的类型和它们的值.

（1）2＊6＊J/I

（2）（3＊6.5）＊2/3＋I

（3）1/2＊I＊A－J

（4）I＊2＊＊（1/2）－A－J

（5）I＊＊2＊8/5＋0.6－X

（6）J＊（X＊2）/（－I＋J）

5. 银行存款中，计算本利和的公式为：

$$P1=P(1+R)^N$$

P 为存款数，R 为年利率，N 为年数，P1 为 N 年后本金和利息之和. 今使 P＝1000，R＝0.05，N＝3，请编程序求三年后本利和，只要取到小数点后二位，第三位四舍五入.

6. 制造一根实心圆铁杆，若圆的直径由 2 厘米增加到 4 厘米，求单位长度要增加多少体积的铁（以立方厘米为单位）？它的质量为多少克（铁的比重为 7.8 克/厘米）？请编写程序.

7. 请编程序，将华氏温度转换为摄氏温度和绝对温度，转换公式分别为：

$$C=\frac{5}{9}\times(F-32)\quad\text{（摄氏温度）}$$

$$K=273.16+C\quad\text{（绝对温度）}$$

已知某物的温度低于 32℉，高于 －459℉，求其相应的摄氏温度（低于 0℃）和绝对温度（高于 0°K）. 只要求取一位小数，对第二位小数四舍五入（请注意，对负数四舍五入的方法）.

8. 已知三角形的二边 a 和 b 和夹角 α，求第三边的长以及另外两个角. 已知 a＝12，b＝20，α＝85°. 请编程序，要求角度以度来表示.

第四章　输入和输出

§4.1　输入和输出的概念

计算机运行一个程序，需要原始数据，运行程序的过程就是对数据进行加工处理的过程，得到的结果也是数据. 向计算机提供原始数据，可以在程序中用赋值语句直接提供，也可以用 DATA 语句提供（见第七章），但当数据量很大时，这种方式是很不方便的，会使程序增长，而且当每次运行时数据有变化需要反复修改程序. 因此，为了使程序具有通用性，常使程序和数据分开，在每次运行程序时，由计算机外面临时将数据送入计算机内，尤其在数据量大的情况下更为如此.

所谓输入输出是以计算机内存为主体而言的. 将外部介质上的数据通过计算机的输入设备送到计算机内存称为输入. 从计算机内存将数据通过输出设备送到外部介质上，称为输出. 外部介质可以是卡片、纸带、显示器键盘、显示屏、磁盘、磁带等.

输入输出是 FORTRAN 中一项重要的功能. FORTRAN 77 提供的输入输出的功能是很强的. 它成功地解决了输入输出的标准化问题，使输入数据方便灵活，使输出数据整齐有规律. 它较好地处理了各种类型的数据的输入输出问题，使输入输出的格式多样化. 因此，应当很好地掌握本章内容，在程序中善于利用 FORTRAN 提供的输入输出语句设计出所需的数据格式.

FORTRAN 77 用于输入输出的语句为：

（1）输入. 用 READ 语句.

（2）输出. 用 PRINT 语句和 WRITE 语句.

输入输出有三种格式：

（1）表控格式（又称自由格式）输入输出.

（2）有格式的输入输出. 按用户所要求的格式组织输出数据.

（3）无格式的输入输出. 即以二进制形式输入和输出. 这种方式只适用于计算机内存与磁盘磁带之间交换数据. 将内存中以二进制形式存放的数据不加转换地输出到磁盘或磁带上，以及将它们从磁盘或磁带上直接读回内存.

在本章以及以后的几章中，只讨论打印输出，即以打印机为输出设备. 显示输出（以终端显示器为输出设备）的概念与用法与打印输出基本上是相同的. 在不少微型计算机系统上，用一个打印语句可以使数据同时输出到显示器和打印机上. 因此在本书中一律以打印输出代表打印和显示. 在输入方面，我们也只讨论键盘输入（从终端显示器的键盘上敲入数据）和卡片输入. 虽然它们二者是从不同的输入设备输入数据的，但从本质上来看，它们是类似的，从键盘上敲入一组字符并按回车键，这叫做输入一个“记录”，相当于一张卡片信息被输到计算机内. 由于用卡片输

入比较容易说明所输入字符的位置，因此，有时我们以卡片输入来说明问题，读者是很容易举一反三的.

打印一行称为输出一个记录. 在输出时是以记录为单位的，即一次输出一行. 同样，每次输入的也是一个记录（例如一张卡片），一张卡片上的信息是不能分二次或几次输入的. 在第十二章中将对记录和文件的概念和应用作进一步的叙述.

在 FORTRAN 中，要进行输入或输出，需要确定以下几个因素：

（1）在什么外部设备上输入或输出.

（2）用什么格式输入或输出.

（3）输入或输出哪些量.

在下面几节中将要解决这个问题.

§4.2 表控输入

表控输入又称自由格式输入，是最简单的一种输入数据的方式. 例如：

```
READ*,A,B,C
```

就是一个自由格式输入的语句，“*”号表示自由格式，输出设备虽然未明显地指出来，但这种形式的 READ 语句隐含指定输入设备为“系统所预先指定的设备”，在一般微型机系统上指定的是终端显示器. 在有些大、中型计算机系统上则隐含指定为读卡机. 该语句要求输入三个实型数，分别赋给三个实型变量 A，B，C.

如果采用键盘输入，可以从键盘上输入以下信息：

```
108.6,-0.37,1.6E2↵
```

数据之间用逗号相隔，也可以用空格（一个空格和多个空格作用相同）相隔. 如：

```
108.6␣-0.37␣␣1.6E2↵
```

作用和上面的相同. 这两者的作用都是使 A 的值等于 108.6，B 的值等于 -0.37，C 的值等于 1.6×10^2.

需要说明的是：

（1）输入的数据应当和 READ 语句中相应的变量类型一致. 如：

```
READ *,I,A,K,X
```

按 I-N 隐含规则，I 和 K 是整型，A 和 X 为实型. 输入以下的数据是正确的：

```
12,18.6,-23,0.6
```

而输入以下数据是不正确的：

```
12.3,18,-5.6,300
```

因为 I，K 变量为整型，需要整型数据，而输入给它们的是实型数. 同样，A 和 X 要求的是实数.

（2）输入数据的个数应当和 READ 语句中变量的个数相等. 如果输入的一个记录中的数据个数少于所要求的个数，则 READ 语句从下一个记录中读数，直到满足所需的数据个数为止.

如：

```
READ ＊,A,B,C,D,E,F
```

以下几种输入数据的方法都是可以的：

① 12.3,246.5,32.6，-8.5,0.63,7.26↵（或一张卡片）

② 12.3,246.5,32.6↵

-8.5,0.63,7.26↵（或用二张卡片）

③ 12.3↵

246.5,32.6，-8.5↵

0.63↵

7.26↵（或用四张卡片）

在第①种情况中，一个记录提供了READ语句所需的全部数据.第②种情况中，第一个记录只提供了三个数据，不满足所需的数据个数，计算机等待从键盘再输入下一个记录，输入二个记录后，READ语句执行完毕.第③种情况是，直到第四个记录输入后，READ语句才执行完毕.

如果一个记录中数据的个数多于READ语句中变量的个数，则多余的数据不起作用.例如：

```
READ ＊,I,J,K
```

如果输入数据为：

11,12,13,14,15

则将11,12,13分别输入给I,J,K，而14,15不被读入.

（3）每一个READ语句都从一个新的记录开始读数.换句话说，不能从一个记录的中间开始读数.例如：

```
READ ＊,I,J,K
READ ＊,N,M,L
```

如果输入以下几个记录：

2,4↵　（第一个记录）

6,8↵　（第二个记录）

10,12↵　（第三个记录）

14,16↵　（第四个记录）

得到的结果并不是I=2,J=4,K=6,N=8,M=10,L=12.在输入第一个记录后，变量K仍未得到值，乃从第二个记录中继续读数，将6读到K中.此时第一个READ语句已执行完毕，接着应执行第二个READ语句.但它要从一个新的记录读数，因此，就从第三个记录的第一个数据开始读数，将10,12,14分别送给变量N,M,L.请特别注意，第二个记录中的“8”和第四个记录中的“16”都未被读入，不起作用.

（4）在输入数据时，用“/”表示数据结束，READ语句不再继续读数.如：

```
READ ＊,A,B,C
```

如输入：

1.5,2.5/3.6↵

在将1.5和2.5读到A和B以后，遇到斜杠，表示不再输入数据给READ语句的变量.因此C未被赋值(C仍保留原来的值).因此，可以在输入数据时，用斜杠来控制对某些变量是否赋之以新值.例如可以只给一个变量赋以新值，而对其他二个变量不赋新值.甚至可以连一个变量都不赋值.在一般情况下可以不必使用"/".

(5) 如果READ语句中有几个连续的变量需要赋予同一个值，可以用重复因子r，表示一数据重复出现r次.例如，对上面的READ语句，可以用下面形式的输入：

3*1.5↵

它等效于：

1.5,1.5,1.5↵

如果输入：

3*␣↵

即输入三个空值，A，B，C三个变量未被输入新的值.如果A，B，C原已有值，则它不改变该值.

(6) 由于空格是分隔两个数据的符号，因此在一个数据的中间不能插入空格.如对以下语句：

```
READ *,I,J,K
```

如果输入：

12␣34, ␣56␣78␣,46

I的值并不等于1234，J的值也不等于5678，K不等于46.而是I等于12，J等于34，K等于56.在数据之前和之后的空格(例如56之前和78之后的空格)不起作用.

表控输入使用比较方便，是FORTRAN 77的改进(FORTRAN 66中无此功能)，它不需程序设计者规定每个输入数据的格式(例如每个数据占多少列，有几位小数等)，因而容易掌握，不易出错，建议初学者尽量使用这种自由格式的输入方式.

有的微机系统不接受这种READ语句，可改用READ(*,*)形式.

§4.3 表控输出

所谓表控输出，指的是由计算机系统隐含指定输出数据的格式.例如，用这种方式打印三个实数，系统会自动为每一个数据留出若干列的位置，并确定该数小数的位数."表控输出"是"表控格式"输出(list directed format)的简称.

可以用两种语句来实现表控输出：

(1) PRINT *,输出表列

(2) WRITE(*,*)输出表列

分述如下.

4.3.1 用PRINT语句实现表控输出

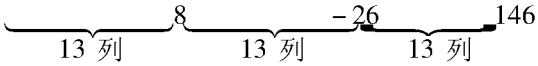
这是最常用而且最简单的一种输出语句.实际上我们已在前几章中用过这种语句了.今再对

它作进一步的说明.

PRINT 语句的输出设备，由计算机系统隐含指定. 每个计算机系统都事先指定了它的隐含输出设备. 一般微型计算机隐含指定为终端显示器或打印机. 因此

```
PRINT *,I,J,K
```

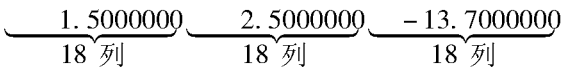
就是将变量 I,J,K 的值打印(或显示)出来，PRINT 语句中的“ * ”表示“表控方式”，即“按系统隐含指定的格式输出”。每一种计算机系统都是先规定好每一种类型的数据在输出时所占的宽度(列数)和格式. 例如在我们所用的计算机系统中，在输出整数时，每个整数占 13 列的位置，数字打印在该范围内的右端. 如果 I,J,K 的值分别为 8，-26,146，则打印结果为：



我们所用的系统在输出实数时，为每一实数留 18 列宽度，打印出 7 位小数. 如：

```
PRINT*,A,B,C
```

若 A,B,C 的值分别为 1.5,2.6，-13.7，打印结果如下：

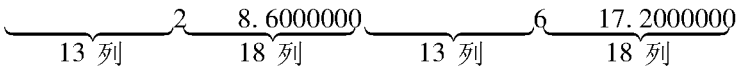


系统会根据数的大小决定按小数形式或指数形式输出(各系统的具体规定不同，读者自己上机试一下即可).

在 PRINT 语句中可以混合使用不同类型的变量，也可以输出表达式的值. 如：

```
PRINT *,I,T,2*3,I*T
```

若 I=2,T=8.6，则打印结果为：



如果输出的记录在一行内打印不下，会自动换行再打印，直到把全部输出的数据打印完为止.

请注意，不同的计算机系统的表控输出有不同的规定. 例如有的规定每个整型量的输出占 10 列，实型量的输出占 16 列. 也有的编译系统则不规定每个整数和实数输出时所占的固定宽度，只打印数据的实际长度，并在两个数据之间空一个或几个空格. 不同的计算机系统之间差别甚大. 读者不必死记，只需上机试验一下即可.

如果在 PRINT 语句中不出现输出表列，如：

```
PRINT *
```

其作用是打印出一个“空白行”，可以利用它实现“隔行打印”等.

有的微型计算机上所配置的 FORTRAN 77 是“子集”，不具备 PRINT 语句(例如 IBM PC 的 FORTRAN 77)，可以用 WRITE 语句代替 PRINT 语句. 以下两个语句作用相同.

```
PRINT *,A,B,I
WRITE(*,*)A,B,I
```

4.3.2 用 WRITE 语句实现表控输出

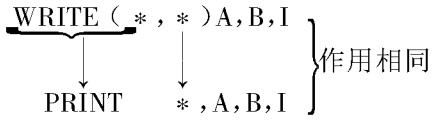
用 PRINT 语句只能在打印机上输出，而有时人们往往希望将数据输出到其他外部介质上（如磁带、磁盘）. WRITE 语句既可用来将数据输出在打印纸上，也可用来输出到计算机所用的任何一种外部介质上. 它的用途比 PRINT 语句广泛，是最有用的 FORTRAN 语句之一.

用 WRITE 语句实现表控输出的方法和用 PRINT 语句实现表控输出的方法类似的.

例如：

```
WRITE( *, * )A,B,I
```

就是表控输出语句. WRITE 语句中括弧内有两个符号，第一个指出输出的设备. 第二个指出输出的格式.“ * ”表示采用“系统隐含”的方式. 第一个“ * ”表示在系统隐含指定的设备（一般为打印机）输出. 因此 WRITE 和第一个星号的作用就相当于 PRINT. 第二个星号表示按系统隐含指定的格式（表控格式）输出. 第二个“ * ”号和上一节中 PRINT 语句中的“ * ”作用相同. 因此



除了可以用“ * ”指出输出的设备以外，还可以用“设备号”来指定输出设备. 每一个计算机系统都与一个或多个输出设备相连，对每一个设备都分配一个设备号. 不少系统以设备号“6”代表打印机或显示器. 因此

```
WRITE (6, * ) A,B,I
```

的作用是：在打印机（或显示屏）上以表控格式输出 A，B 和 I. 它的作用和上面两个语句相同.

如果不想在打印机上输出，只要改变设备号即可. 每种计算机系统的输出设备所对应的设备号是不同的，请务必弄清楚本计算机系统设备的代号. 关于向磁盘（或磁带）输出数据的方法请见第十二章.

“设备号”又称“通道号”或“文件号”.

有了表控输入输出语句，就可以用来编写程序了.

例　求复利. 在银行存款 p 元. 按年利率 r% 计算利息. 可以任意指定每年结算的次数 m，并指定年数 n. 例如一年结算二次，共五年，则 m = 2，n = 5. 本利和的公式为：

$$A = p\left(1 + \frac{r}{100\ m}\right)^{m \cdot n}$$

按此编写程序. 先读入 p，r，m 和 n 的值，然后打印出 p，r，m，n 和 A 的值.

```
C COMPOUND INTEREST PROGRAM
      READ ( *, * ) PRINCI, RATE,TIMES,YEARS
      WRITE ( *, * )'PRINCIPAL $',PRINCI
      WRITE ( *, * )'INTEREST RATE',RATE,'PERCENT'
      WRITE ( *, * )'COMPOUNDED',TIMES,' TIMES PER YEAR'
```

```
      WRITE( * , * )'FOR',YEARS,'YEARS'
      AMOUNT = PRINCI * (1 + RATE/(100 * TIMES)) * *
     #     (TIMES * YEARS)
      WRITE( * , * )'YIELDS A TOTAL AMOUNT OF $',AMOUNT
      END
```

在程序中尽量使变量名能代表一定的含义.因此以 PRINCI(PRINCIPAL 的前 6 个字母)代表“本金”p,以 RATE 代表利率 r,TIMES 代表一年结算的次数 n,以 YEARS 代表总年数 m 以 AMOUNT 代表本利和 A.

程序运行结果为:

```
100.0,8.0,2.0,5.0↵(输入 p=100,r=8,n=2,m=5)
PRINCIPAL $                    100.0000000
INTEREST RATE                   8.0000000 PERCENT
COMPOUNDED                      2.0000000 TIMES PER YEAR
FOR                          5.0000000 YEARS
YIELDS A TOTAL AMOUNT OF $          148.0244000
```

再运行一次,输入 p=1000,r=7.5,n=5,m=10.

```
1000.0,7.5,5.0,10.0↵
PRINCIPAL $           1000.0000000
INTEREST RATE                7.5000000 PERCENT
COMPOUNDDE                   5.0000000 TIMES PER YEAR
FOR                 10.0000000 YEARS
YIELDS A TOTAL AMOUNT OF $          2105.2410000
```

用表控输出虽然方便,但当输出的数据量比较大时,打印出来的各个数据往往不大整齐,不同类型的数据上下两行之间不易对齐,这给人们检查打印结果造成不便.因此它只用于少量数据的输出.大量数据的输出常用格式输出.

§4.4 格式输出

FORTRAN 包括多种类型的数据.人们希望对各种类型的数据都能按要求实现输入输出.FORTRAN 的一个突出的优点就在于成功地解决了输入输出的标准化问题,使输入输出的形式多样化,可以根据用户需要输出各种不同格式的数据.

由于格式输出比格式输入容易理解些,所以我们准备先介绍格式输出和格式说明符,然后再介绍格式输入.

4.4.1 最简单的格式输出语句

简单的格式输出一般由一个 WRITE 语句和一个 FORMAT 语句组成.WRITE 语句指出输出设备、格式语句的标号和输出表列.FORMAT 语句(格式语句)提供具体的格式信息.

假如我们要将整型变量 L,J,K 的值输出,并希望在打印时每个数据分别占 6 格、7 格和 8 格

宽度. 可以写成：

```
      WRITE ( * ,100) L,J,K
      |_____________|
100   FORMAT (1X,I6,I7,I8)
```

WRITE 语句中，括弧内的第一个“ * ”的作用同前述，即指定数据从隐含指定的输出设备输出. 括弧内第二项不再是“ * ”，而是数字“100”. 前已指出，括弧中的第二项的作用是用来指定输入输出格式的. 这个“100”是一个格式语句（FORMAT 语句）的标号，它指出：在输出时按标号为 100 的 FORMAT 语句提供的“格式说明”进行输出.

在本例中，FORMAT 语句提供的“格式说明”为“1X，I6，I7，I8”. 其中“1X”的作用是使在打印数据时先换一行再开始打印本行内容. 在一般的打印输出中，FORMAT 语句中括弧内的第一项多用“1X”. 我们可以先承认它、记住它. 在本章的稍后我们再详细介绍它的作用.

“I6，I7，I8”是为输出的变量 L，J，K 提供格式说明的. 在输出时，一行（即一个记录）内要容纳若干个数据，每个数据的范围称为字段（field）. I6，I7，I8 是三个“字符描述符”（field descriptor）或称“编辑描述符”（edit Descriptor）. 它的作用是对每个字段的数据规定输入输出的格式.“I”是整型编辑描述的符号.“I”后面跟一个数，指出字段的宽度，“I6”表示输出的整型变量占 6 个字符的宽度. 如果 L = 10，J = －123，K = 24，则上面的格式输出结果为：

```
␣␣␣␣10␣␣␣－123␣␣␣␣24
6 列    7 列    6 列
```

整型数据的输出必须用整型编辑符 I，同样，实型数据的输出应当用实型编辑符.

例如，用以下语句可以输出实型变量 A，B，C 的值.

```
      WRITE ( * ,200) A,B,C
200 FORMAT (1X,F10.2,F15.4,F6.3)
```

若 A = 3.6，B = 5.36，C = －2.1，输出形式为：

```
␣␣␣3.60␣␣␣␣␣5.3600␣␣␣␣－2.100
10 列      15 列      6 列
```

其中“F”是实型编辑符，F10.2 表示打印的实数共占 10 列，其中小数点后有二位数字. 其余的类似.

由此可知，格式输出的关键是如何指定输出的格式. FORMAT 语句中括弧内的内容就是“格式说明”. FORTRAN 提供了多种编辑描述符，以适应各种不同类型数据的输入输出，以及完成其他的输入输出功能. 之所以称它们为编辑描述符，是因为它们对数据的格式进行描述，输出时将内存中的数据按此格式进行“编辑加工”，然后打印出来.

4.4.2 I 编辑符

I 编辑符专用于整型数据的输出，它的一般形式是：

（1）Iw

I 代表 Integer（整数），w 表示字段宽度，例如 I5 表示该整型数应占的列数为 5 列. 如果该整数的实际位数（包括符号）不足 w 位，则左面补以空格. 如果多于 w，则无法正确表示出该数，以 w

个“ * ”表示. 如：

```
        WRITE ( * ,100) I,J,N
100     FORMAT (1X,I5,I6,I4)
```

若 I = 12345，J = -24，K = 24689，则打印结果为：

12345 -24****
5 列 6 列 4 列

K 实际上应占 5 列，但 FORMAT 语句中的格式编辑符“I4”只提供 4 列位置，无法容纳 24689 达五位数字，乃在该范围内全部打印“ * ”号，表示“字段宽度不够”出错.

（2）Iw. m

这是 FORTRAN 77 增加的格式. w 的含义同前，m 表示需要打印的数字的最少位数. 例如：

I 10. 5

表示字段宽度为 10，其中至少应当有 4 位为数字.

```
        WRITE ( * ,100) M,M
100     FORMAT (1X,I10,I10.4)
```

如果 M 的值为 24，则打印结果为：

␣␣␣␣␣␣␣␣24 ␣␣␣␣␣␣0024
10 列 10 列

M 只有二位整数，但由于“I10. 4”要求至少应有 4 位数字，因此在 24 之前补了二个“0”. 而用“I10”描述的，只打印出 24.

如果输出的整数实际位数超过 m 位，则按实际长度打印. 如 M = 26834，则打印结果为：

␣␣␣␣␣26834 ␣␣␣␣␣26834
10 列 10 列

二者相同.

注意 m 位数字不包括负号. 如上例中 M = -268，则打印结果为：

␣␣␣␣␣␣-268 ␣␣␣␣-0268

即保证有 m 位（今为 4 位）数字.

显然 m≤w，如果输出的是负数，则 m 应小于 w. 在设计程序时，应使 w 大于实际输出的位数，例如输出的数是 5 位整数，而用 I3，显然会出现“字段宽度不够”. 应该对输出的数的大小事先估计其位数，如果位数为 b，则应使 w≥b，如果输出负数，则使 w≥b + 1. 但事实上不可能准确估计每一个输出量的大小，可以使 w 大于计算机允许的最大整数的位数. 例如有的微型计算机 FORTRAN 允许的整数范围为 -32768 ~ 32767，五位数字，因此 w 选 6，7，8 或更大一些均可. 有的计算机系统允许有十位整数，因此应使 w≥12，如 I12，I15 等.

4.4.3 F 编辑符

F 编辑符用于实型量的输出，它使输出的数据以“小数形式的实数”表示. F 是 Float（浮点

数）的缩写.

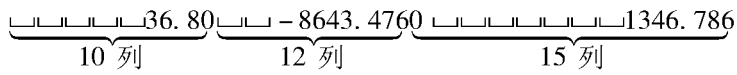
F 编辑符的一般形式为：

Fw.d

其中 w 为字段宽度，d 为该数的小数位数. 例如 F12.4 表示其数据输出时共占 12 列，其中有 4 位为小数部分，例如：

```
      WRITE(*,200) T,U,X
200   FORMAT(1X,F10.2,F12.4,F15.3)
```

若 T=36.8，U=-8643.476，X=1346.7856，则打印结果如下：



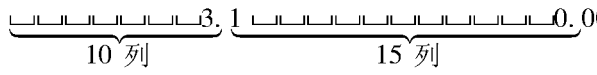
可以看到 T,U,X 三个输出量分别使用 F10.2，F12.4 和 F15.3 三个编辑描述符. T 的值为 36.8，由于 F10.2 要求有二位小数，因此在小数点后第二位补一个零. 同样 U 在打印时在小数点后第四位补一个零. X 的值为 1346.7856，而 F15.3 规定只打印出三位小数，因此第四位小数无法打印出来，FORTRAN 规定对截断后一位采取四舍五入处理，因此打印出来的是 1346.786.

归纳起来，如果输出的数值的小数部分位数少于 Fw.d 中的 d，则应以零补齐 d 位小数. 如果实际的小数部分位数大于 d，则对小数后第 d+1 位产生截断，并按四舍五入处理.

应该恰当选择 w 和 d 的值，如果要输出的数值比较小（如 3.1415926535，0.001245），若 d 选小了，就会丢掉一些有效数字而产生误差. 这就是“小数印丢”. 例如：

```
      WRITE(*,100) X,Y
100   FORMAT(1X,F10.1,F15.2)
```

若 X=3.14159265，Y=0.001245，则打印结果为：



但如果要输出的数值比较大，而选 w 不够大或 d 太大时. 会出现“字段宽度不够”. 如对 24680134.1，用 F10.6 输出. 为保证 6 位小数应该是：24680134.100000. 但规定了 w=10，只能共占 10 个字符宽度，今却要求 15 位，显然不够，整个字段以“*”充满之：

****

这就是“大数印错”.

在编写程序时，应使 w≥b+d+1. 其中 b 为数的整数部分位数，d 为小数部分位数，还有 1 位小数. 例如，想打印 1357.95，w 应不小于 4+2+1=7. 可选 F7.2 或 F8.2 等. 如果打印的是负数，则应使 w≥b+d+2. 因为还应加一个负号的位置. 一般总把 w 适当选大一些，如 F15.2，F16.3，F16.6 等. d 的值根据需要选择，但不要盲目地选大的值（如 F12.8，F15.9 等）. 应当注意 d 若偏小，数据还能打印出来，只是产生一些误差（低位被截去），而若 w 偏小，则整个数据无法打印出来，而以全“*”号标志出错，FORTRAN 不采取截去高位而打印低位的办法（如对 2468.26 只打印 468.26），因为这样的打印结果是无意义的，而且是有害的，它给人们一个完全错误的结果.

下表是用 F 编辑描述符输出的结果.

要输出的值	编辑描述符	打印结果	说　　明
6.5	F8.2	␣␣␣␣6.50	小数点后第二位补零,数字前补空格
867.65783	F8.3	␣8 6 7.658	第四位小数四舍五入
-6721.83274	F9.4	*** * * * * * *	需要 10 位才放得下,字段宽度不够
0.0008376	F10.4	␣␣␣␣0.0008	第四位小数以后的被截去
1683.246	F6.2	** * * * *	需要 7 位才能放下,字段宽度不够

用 F 编辑描述符来指定输出格式的好处是打印出来的数据的形式比较直观,符合人们的习惯(以小数形式表示),但缺点是往往难以确切估计数的大小,从而选择合适的 w 和 d 的值.

4.4.4　E 编辑符

E 编辑符也是用于实型数据的输出,但它输出的形式是以指数形式表示的实数. 它的一般形式为:

Ew.d

E 是 Exponent(指数)的第一个字母,用它来表示按指数形式进行输出. 例如:

```
      WRITE(*,100) X,Y
100   FORMAT(1X,E12.4,E15.6)
```

若 X = 8.52, Y = 3014.159, 输出的格式为:

$$\underbrace{\text{␣␣0.8520E␣01}}_{12\text{ 列}}\underbrace{\text{␣␣␣0.301416E␣04}}_{15\text{ 列}}$$

可以看出:输出的是以指数形式表示的实数,在字段中指数部分必占四列,其中“E”占一列,符号占一列(正号一般不印出),指数占两列. Ew.d 中的 d 指出数据的数值部分(即 E 前面的部分)中小数的位数. 一律以标准化的指数形式表示(即小数点前无非零整数,小数后第一位为非零数字). 上例中 $X = 8.52 = 0.852 \times 10^{1}$,而 E12.4 要求打印四位小数,因此输出为:␣␣0.8520E␣01. $Y = 3014.159 = 0.3014159 \times 10^{4}$,而 E15.6 要求打印 6 位小数,故对第七位小数四舍五入,打印结果为␣␣␣0.301416E␣04. 数据一律在字段的右端,左面无数字和符号的位置上填以空格.

w 的值应该不小于 d+6. 因为有 d 位小数,再加一个“小数点”,小数点前有一个零,还有四列是指数部分,所以共为 d+6 列. 如果输出的是负数,则应 w≥d+7. 因此想以指数形式输出一个数 123.4568,希望小数位数为 6 位,则 w 应≥6+6=12,可以选 E12.6 或 E13.6 等.

用 E 编辑符可以输出范围很大的数,且可避免用 F 编辑符时容易出现的“大数印错,小数印丢”的缺点. 同时又可以保证有必要的有效位数的数字. 如果用 E15.6 则可保证有 6 位有效数字. 如果所用的 FORTRAN 提供的常数有 9 位有效数字的话,则应该使 d=9,以取得最多的有效数字,此时 w 的值应取 9+7=16,即 E16.9. 这是保险的方法,不论对多大的数都适用。例如,当 $A = 2.6 \times 10^{39}$, $B = -6.9283267 \times 10^{-12}$时,用下面输出语句:

```
      WRITE(*,150) A,B
150   FORMAT(1X,E16.9,E16.9)
```

输出结果为：

```
␣0.260000000E␣40 -0.692832670E-11
 16列             16列
```

可见，很大的数和很小的数用指数形式输出是没有什么问题的. 以上二数虽能正确打印出来，但二者紧凑地连在一起，难以分辨. 可以用下一段介绍的“X”编辑符插入一些空格，或者使 w 取大一些的值，如取 E18.9 等.

下表是用 E 编辑符输出的情况.

要输出的值	编辑描述符	打印结果	说　　明
728.653	E10.2	␣␣0.73E␣03	先化成 0.728653×10^{3}
0.00000000124	E12.4	␣␣0.1240E-08	先化成 0.124×10^{-8}
1864.323×10^{13}	E13.6	␣0.186432E␣04	先化成 0.1864323×10^{4}
72.8×10^{6}	E12.7	**********	w 应≥d+7=14，字段宽度不够

有的计算机系统允许有较大的实数范围，例如有的可以用到 10^{337}. 这时指数用二位整数就不够用了. FORTRAN 77 提供一种扩充的 E 编辑符，可以输出三位或四位指数. 这种形式的 E 编辑符为：

Ew.dEe

其中 e 为指数的位数. 如想输出 1.6×10^{123}，可以用 E18.9E3，它表示字段宽度为 18，数值部分的小数位数为 9 位，指数位数为三位. 输出结果为：

```
␣␣0.160000000E␣124
  2列  9列     5列
```

实际上，上面 Ew.d 形式的编辑描述符是 Ew.dEe 的一种特殊形式，只是隐含将 e 的值定为 2.

有些计算机系统将 e 的隐含值定为 3，如果写 E12.4，输出时打印出三位指数，如：

```
␣0.1230E␣002
 2列 4列 5列
```

说明：

（1）实型数据不能用 I 型编辑符输出. 同样，整型数据不能用 F 或 E 编辑符. 从计算机内存中输出数据时，按照编辑符的类型进行数据形式的转换. 例如，整数在内存中以定点二进制形式存放，如用 I 编辑符输出，则先把数据转换成十进制的整型数，然后输出.

（2）一个实型数据可以用 F 编辑符输出，也可以用 E 编辑符输出. 在内存中实数一律是用指数形式存放的（浮点数形式），如果用 F 编辑符，则将内存中的数据先转换成十进制的小数形式，然后输出，如用 E 编辑符，则先化成十进制的指数形式然后输出. 程序设计者可以任选 F 型或 E 型编辑符进行输出. 例如，若 A=326.76，有：

```
      WRITE(*,100) A,A
```

```
100     FORMAT(1X,F10.4,E12.4)
```

输出结果为：

```
␣␣326.7600␣␣0.3268E␣03
 10列      12列
```

4.4.5 X编辑符

为了避免相邻的两个数据紧连在一起，可以用X编辑符在数据之间插入一些空格.它的一般形式为：

nX

n是希望插入的空格个数.如：

```
        WRITE(*,100) I,J,A
100     FORMAT(1X,I3,2X,I4,3X,F6.3)
```

若I=108，J=1832，A=67.82，打印结果为：

```
108␣␣1832␣␣␣67.820
 I     J      A
```

在执行WRITE语句时，按照标号为100的FORMAT语句进行格式控制：①“1X”的作用使打印机“当前位置”处在本打印行的第一个位置.② 按I3格式打印变量I的值108，打印完后打印机头位置在第4列，准备输出下一个量.③ FORMAT中的“2X”使打印机头位置从“当前位置”右移2列，到第6列.④ 打印变量J的值(按I4格式)，J的值为1832占4列，打印完后打印机头移到第10列.⑤“3X”又使打印机头右移3列，到13列.⑥ 按F6.3格式输出A的值67.820，打印完后打印机头位置在第19列.以上步骤见下图示意：

```
列数:1 2 3 4  5 6 7 8 9 0 1  2 3 4 5 6 7 8 9 0
打印内容:1 0 8␣␣1 8 3 2␣␣␣6 7 . 8 2 0
          ↑  ↑  ↑    ↑    ↑      ↑
          ①  ②  ③    ④    ⑤      ⑥
```

FORTRAN 77允许nX中的n取负值，它表示从当前位置向左移动若干列定位.例如：

```
        WRITE(*,200) X,T,I
200     FORMAT(1X,F6.2,15X,F6.1,-15X,I4)
```

输出结果为：

```
列数:1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8
内容:1 2 3 . 5 6          2 4 5 7          2 4 5 6 . 3
       ↑          ↑          ↑    ↑          ↑          ↑
       ①          ②          ⑤    ⑥          ③          ④
```

执行过程可以这样理解：有一指针指向“当前位置”.① 开始时指针指向本行第1列；② 按F6.2格式打印变量X的值123.56，打印完后指针位置在第7列；③“15X”使指针右移15列，指向第22列；④ 按F6.1格式打印变量T的值2456.3，打印完后指针位置在28列；⑤“-15X”使指针左移15列，指向第13列，⑥ 按I4输出变量I的值2457，最后定位于17列.

有的微机上所用的FORTRAN 77不能使用n为负值的X编辑符.

4.4.6 H 编辑符

用来输出字符常数. 它的一般形式为：

nH〈字符序列〉

字符序列是由若干字符组成的，字符的总个数为 n.

例如：

```
8HABCDE␣FG
```

在执行相应的 WRITE 语句时，将输出“ABCDE␣FG”八个字符.

假如有以下语句：

```
      WRITE(*,100) P,R,Y,A
100   FORMAT(1X,10HPRINCIPAL =,F6.2,2X,5HRATE =,F4.2,2X,
     #  6HYEARS =,F4.1,2X,7HAMOUNT =,F10.2)
```

打印结果为：

```
PRINCIPAL = 100.00 ␣␣RATE = 0.05 ␣␣YEARS = ␣2.0 ␣␣AMOUNT = ␣␣␣␣110.25
```

请注意输出字符常数的位置. 在 FORMAT 语句中，在“1X”之后出现了 H 编辑符，它并不是变量 P 或 R 或 Y，A 的字段描述符，而是要求输出 11 个字符(PRINCIPAL =)，因此在本行的第一列开始打印“PRINCIPAL = ”，FORMAT 语句中的下一个编辑符是 F6.2，它是实型编辑符，对应 WRITE 语句中第一个实型变量 P. 在打印出 P 的值(100.00)后，遇“2X”，插入两个空格，再输出五个字符“RATE = ”，然后再按 F4.2 编辑符输出变量 R 的值(0.05)，再空两格，……. 直到按 F10.2 格式输出最后一个变量 A，结束本记录(行)的输出.

在用 H 编辑符时，务请注意字符常数的字符数不要数错，否则会出现错误，如想输出 A 的值，用下面格式语句：

```
100   FORMAT(1X,3HA =,F6.2)
                 3个字符
```

本来希望能打印出：

```
A = 124.57
```

这样的结果来. 但编译时却出错，原因是用了“3HA = ”，而本应用“2HA = ”. 由于写了“3H”，系统便按此要求将“H”后面的三个字符“A =，”认为是需要打印出来的字符常数. 而“，”本来是两个编辑描述符之间的分隔符，现在却被挪作他用了，因此两个编辑描述符间缺少了一个必须存在的逗号，故出现“语法错误”. 如果写成“1HA = ”，就只将 H 后面的一个字符 A 作为需要打印的字符常数，而把“ = ”排除在外，编译系统无法辨别此“ = ”的作用，按出错处理.

H 编辑符不要求 WRITE 语句中有相应的输出项，如：

```
      WRITE(*,200)
200   FORMAT(1X,7HBEIJING)
```

WRITE 语句中无任何输出项，执行的结果是打印：

```
BEIJING
```

用 nHccc…c 形式表示的字符常数称为 Hollerith 常数，Hollerith 是一个人名. 符号“H”就来源于此.

4.4.7 撇号编辑符

用 H 编辑符必须要求准确地数出“H”后面的字符个数. 如果字符个数很多，很易数错，而造成编译出错，这是很不方便的. FORTRAN 77 提供了功能与 H 编辑符相同而使用方便的撇号编辑符. 它的作用是提供输出的字符常数——撇号内的字符. 下面两个 FORMAT 语句等价：

```
100    FORMAT(1X,13 H HOW ␣ DO ␣ YOU ␣ DO)
100    FORMAT(1X,'HOW ␣ DO ␣ YOU ␣ DO')
```

显然第二个 FORMAT 语句方便明确，不需数字符个数，而且字符常数的起止界限明确. 建议用撇号编辑符而尽量少用 H 编辑符. FORTRAN 77 之所以保留 H 编辑符，仅仅是考虑到与 FORTRAN 66 标准的兼容.

在使用撇号编辑符时，如果欲打印包括撇号的字符常数怎么办？可以用两个连续的撇号作为一个输出字符“''”(撇号). 例如，想打印：

```
THIS  IS  MY  TEACHER'S  BOOK.
```

FORMAT 语句可写为：

```
100    FORMAT(1X,'THIS  IS  MY  TEACHER S  BOOK.')
```

请注意 TEACHER 后面有两个连续的撇号. 在编译时将外层的撇号当作字符常数的起止字符而将里层的两个相邻的撇号当作一个撇号来处理.

4.4.8 重复系数

对 I,F,E 型编辑符(以及以后将介绍的 D,G,L,A 型编辑符，可以在它们前面加一个重复系数，以代表该编辑描述符重复出现若干次. 它们的一般形式为：

rIw, rIw.m, rFw.d, rEw.d, rEw.dEe, …

例如：

```
100    FORMAT(1X,3F6.2,2X,2I6)
```

其作用等同于：

```
100    FORMAT(1X,F6.2,F6.2,F6.2,2X,I6,I6)
                  ↓              ↓
                 3F6.2          2I6
```

重复系数 r 必须是无符号的整数，不能是变量或表达式.

可以用括弧将若干个编辑描述符括起来组成一个编辑描述符组. 对编辑描述符组也可以用重复系数. 例如：

```
150    FORMAT(1X,I6,2X,F6.2,2X,I6,2X,F6.2,2X,F10.6)
```

可以写成：

```
150    FORMAT(1X,2(I6,2X,F6.2,2X),F10.6)
```

撇号和 H 编辑符前不能再加重复系数. 但必要时可以先加一括弧然后再加重复系数. 如:

```
180     FORMAT(1X,20('*'),40('-'),20('*'))
```

将在一行上打印 20 个“ * ”,40 个“ - ”,和 20 个“ * ”,共 80 个符号. 当然也可以改写成:

```
180     FORMAT(1X,20(1H*),40(1H-),20(1H*))
```

4.4.9 纵向走纸控制

在打印输出时,往往根据用户需要使行与行之间按一定规律相间隔(如一行接一行打印,或一行空一行打印等等). FORTRAN 77 提供了用户可以控制纵向走纸的功能.

FORTRAN 是这样处理的:将输出记录的第一个字符作为“纵向走纸控制符”,它的作用如下表:

纵向走纸控制符	打印前纵向走纸控制
空　　格	走纸一行
0	走纸两行
1	走纸到下一页的第一行
+	不走纸,从本行开头重新叠打
其他字符	大多数编译系统按走纸一行处理

假如我们在第一个字符处放一个“空格”,则系统就会在打印本行之前先走纸一行.

需要说明的是:

(1) 在用一般的 FORMAT 语句提供的格式说明进行输出时,打印完一行后是不换行的,打印的“当前位置”在最后一个字符的后面. 如:

```
        WRITE(*,100) A
100     FORMAT(1X,'A=',F6.2)
```

输出结果:

```
A = 346.52
      ↑
    当前位置
```

因此在打印下一行之前必须先使“当前打印位置”移到下一行的第一个字符位置处. 也就是使打印机向前走纸一行,并使打印机从行的开头处开始打印. 如果需要一行隔一行地打印,则应使走纸两行. 这就需要在输出的记录的第一个字符处放一个“1”.

(2) 打印记录的第一个字符只被系统当作纵向控制符处理,而不将此字符输出,通俗地说,这个字符被“吃掉”.

假如有一记录在内存中为:

```
┌──────────────────────┐
│ ␣1863.65␣␣326        │
└──────────────────────┘
  ↑
第一个字符
```

记录的第一个字符为“空格”,被拿去作纵向走纸控制,在打印本行前走纸一行. 实际上在纸

上打印出来的是从第二个字符开始的全部字符.即:

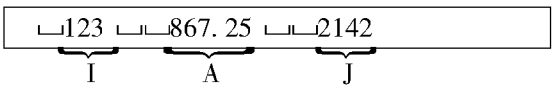
1863.65 ␣␣326

也就是说,打印出来的字符要比内存中输出记录中的字符数少一个.因此,为了要能在纸上打印 1863.65 ␣␣326,应当事先在第一个字符位置处加一个适当的字符(例如,加一个空格或“0”,“1”,“ +”等).这就是我们在 FORMAT 语句中的第一项加一个“1X”的原因.

从 X 编辑符的介绍可以知道,“1X”的作用是产生一个空格.由于这个空格占据输出记录的第一个字符,因此它被当“纵向走纸控制符”,使走纸一行.这个空格并不输出.如果有以下语句:

```
      WRITE(*,100) I,A,J
100   FORMAT(1X,I3,2X,F6.2,2X,I4)
```

在执行 WRITE 语句时,先按 FORMAT 语句规定的格式,在内存中组织输出记录,其形式为:



然后将第一个字符“吃掉”,判断它的作用是“走纸一行”,换行回车后,在打印机上打印出:

123 ␣␣867.25 ␣␣2142

也可以不用“1X”,而用“1H ␣”或′ ␣ ′,作用都是在输出记录第一个字符处放一“空格”字符.

```
100   FORMAT(1X,3I6)
100   FORMAT(1H␣,3I6)
100   FORMAT('␣',3I6)
```

作用完全相同.

(3)如果不在第一个字符处有意识地增加一个字符,则在打印时会“吃掉”本应输出的有用字符,而出现不应有的错误.例如,想输出“BEIJING”七个字符.如果不在 FORMAT 语句中的第一项加“1X”,而只写:

```
100   FORMAT('BEIJING')
```

则系统就将第一个字符“B”当作纵向控制符(按走纸一行处理)而打印出:

EIJING

如果 A = 1873.65,I = 13626 如有以下语句:

```
      WRITE(*,200) A,I
200   FORMAT(F7.2,2X,I5)
```

本想打印出:

1873.65 ␣␣13626

但第一个字符被“吃掉”使纸移动到下一页的第一行,打印出:

873.65 ␣␣13626

与原意不同了.因此请千万注意,一定要事先准备一个供“吃掉”的字符放在记录的第一字符位置.

(4)有的微型计算机系统无分页功能,因此不能用“1”来控制换页.在中型计算机系统上,

可以实现分页，一般以64行为一页，打满64行后自动换页. 也可用“1”控制换页.

（5）如果用“+”作纵向控制符，则不走纸（不换行），只回车，使从本行开头重新打印一行，即两行叠打. 例如：

```
      WRITE(*,150)
      WRITE(*,200)
150   FORMAT('␣===')
200   FORMAT('+///')
```

执行完第一个语句后打印出：

```
===
```

在执行第二个语句时不换行，在原行叠打三个“/”，得：

```
≠≠≠
```

可以巧妙地利用叠打，打印出所需的字符，如在“Y”上叠打“=”，可得“¥”，在“0”上叠打“-”，可得⊖，在“>”上叠打“/”可得“≯”等.

下表是表示纵向走纸控制符的作用的.

FORMAT 语句	内存中输出记录的内容	打印出的信息
FORMAT(1X,I3,2X,I4)	␣112 ␣␣2456	112 ␣␣2456（走纸一行）
FORMAT(I4,3X,I5)	4837 ␣␣␣12387	837 ␣␣␣12387（走纸一行）
FORMAT(F6.2,I3,2X,I4)	123.46246 ␣␣1246	23.46246 ␣␣1246（跳页）
FORMAT(5X,I2,2X,F6.2)	␣␣␣␣␣86 ␣␣246.83	␣␣␣␣␣86 ␣␣246.83（走纸一行）
FORMAT('ABCD',I3)	ABCD876	BCD 876（走纸一行）
FORMAT('+-*/','ABCD')	+-*/ABCD	-*/ABCD（不走纸，叠打在前一行上）
FORMAT(I4.4,2X,I3)	0246 ␣␣␣12	246 ␣␣␣12（走纸两行）

4.4.10 斜杠编辑符

在FORMAT语句中使用斜杠编辑符的作用是：结束正在输出的记录并开始下一个记录的输出. 如：

```
      WRITE(*,300) I,J,K,A,B,C
300   FORMAT(1X,3I5/1X,3F6.2)
```

输出I,J,K的值后遇斜杠，本行结束. 而下面又有“1X”，表示换到下行再打印A,B,C之值. 打印结果为：

```
␣␣328 ␣1246 ␣␣␣12
182.34 ␣␣1.16 ␣25.17
```

共占两行. 如果将斜杠改为逗号，打印结果如何？请读者思考，并比较之.

除了以上几种编辑描述符外，FORTRAN 77还提供了一些其他的编辑描述符（如D,G,A,L,冒号,BN,BZ,T,TR,TC,S,SP,SS等），对其中的几种（D,A,L编辑符）将在以后几章中陆续介绍，而其他一些对初学者来说是用得较少的，因此本书不作介绍. 在今后编制程序时，如果需要用

到它们，可以查阅有关手册即可.

4.4.11 WRITE 语句和 FORMAT 语句的相互作用

输出一个记录的内容是由 WRITE 语句和 FORMAT 语句提供的格式说明（FORMAT 语句中括弧内的内容称“格式说明”）共同作用的结果. 打印出来的信息来源于两个方面：一是 WRITE 语句中输出表列的值；二是 FORMAT 语句中提供的字符常数（由 H 编辑符和撇号编辑符提供的）和空格（由 X 编辑符提供的）. 因此要综合这两个语句，特别是掌握它们之间的互相关系，是十分重要的.

（1）WRITE 语句中输出量的个数和 FORMAT 语句中 I，F，E（还有以后介绍的 D，A，L）编辑描述符的个数可以相等也可以不相等. 如果输出量的个数少于编辑描述符（不包括 H 编辑符、撇号编辑符和 X 编辑符，下同）个数，则多余的编辑描述符不起作用. 如：

```
          WRITE( *,102) A,    I,       K
                        |     |        |
102       FORMAT(1X, F  6.2,I6,2X,I7,I8)
```

在输出完 K 的值后，格式说明中还有一个“I8”编辑符，它不再起作用. 但是如果“I7”后面跟的是 X 型或 H 型（包括撇号编辑符，下同）编辑符，则接着输出 X 型或 H 型编辑符提供的信息，直到遇到非 X 非 H 编辑符为止. 如果上面的 FORMAT 语句改为：

```
102     FORMAT(1X,F6.2,I6,2X,I7,'END',I8,2X)
```

打印结果如下面形式：

```
 128.56 ␣␣1834 ␣␣␣␣␣␣2876END
```

然后遇“I8”而终止输出.“I8”后面的“2X”也不起作用.

（2）如果输出量个数多于编辑描述符个数，即输出表列中还有未输出的元素，而格式说明中的编辑描述符已用完，则重新使用该格式说明，但另产生一个新记录. 如：

```
          WRITE( *,105) A,B,C,D
105       FORMAT(1X,2F6.2)
```

在输出 A，B 的值后，格式说明已用完，遇右括号，乃重复使用此 FORMAT 语句，再输出 C，D，组成一个新记录. 结果如下：

```
246.83186.57 ┐
112.38 18.60 ┘两个记录
```

它相当于用了如下的 FORMAT 语句：

```
105     FORMAT(1X,2F6.2/1X,2F6.2)
```

但是它并不等价于：

```
105     FORMAT(2(1X,2F6.2))
```

因为它将结果全打印在一行上.

如果格式语句改为：

```
105     FORMAT(1X,F6.2)
```

则将打印四行.

（3）如果在格式说明中包含有重复使用的编辑描述组，则当格式说明用完后再重新使用时，只有最后一个描述符组（包括其重复系数）和它右面的编辑符被重复使用. 如：

```
      WRITE ( * ,110) A,B,C,D,E,F,G,H,X
110   FORMAT (1X,2(F6.2,2X),F7.2,2X,2(F8.2,2X),F10.4)
```

在输出完 A，B，C，D，E，F 六个变量的值以后，格式说明已用完，应重复使用格式说明，但此时不从最左面开始，而从最后一个编辑描述组开始重复使用. 打印结果如下：

```
124.89␣␣␣11.94␣␣␣18.79␣␣␣118.62␣␣10182.12␣␣␣124.2014
  6        6        7        8        8          10
1278.66␣␣␣986.80␣␣␣1836.7760
  7        8          10
```

请注意，第二行中的第一个数据只占 7 列，因为该行第一个字符被作为纵向控制符而被“吃掉”.

（4）如果有输出量而格式说明中无相应的编辑描述符（非 X，H 型），则输出永不停止. 如

```
      WRITE ( * ,150) A
150   FORMAT (1X,'A = ')
```

可能程序设计者忘记了在 FORMAT 语句中写“F10.2”，结果将是不终止地输出一个又一个记录，每个记录（每行）打印一个“A = ”. 因此，如果输出表列有一个或一个以上的输出量的话，则格式说明中应至少有一个非 X，H 型的编辑描述符.

（5）在执行格式输出时，系统对输出表列和格式说明同时扫描. 按格式说明中各字段编辑描述符的次序输出各值（如遇重复系数，就按其指定次数进行重复），X 型和 H 型（包括撇号）编辑符不要求输出表中有相应的输出量，而 I，F，E（以及 D，G，L，A 等）编辑描述符则要求必须有相应的输出量才能进行输出. 因此

```
      WRITE ( * ,120) I,A,X
120   FORMAT (1X,'I = ',I6,2X,'A = ',F6.2,2X,'X = ',F6.2,2X,I7)
```

输出的顺序是：①打印“I = ”；②打印 I 的值；③空二格；④打印“A = ”；⑤打印 A 的值；⑥空二格；⑦打印“X = ”；⑧打印 X 的值；⑨空二格. 然后结束本记录的输出. 最后一个“I7”由于没有相应的输出量，故不起作用.

（6）可以有“空格式说明”，如 FORMAT(　　). 用来输出一个空行. 但此时 WRITE 语句中不应有输出量.

（7）遇格式说明的右括号（即 FORMAT 语句中最右面一个括号）或斜杠“/”时，结束本行的打印. 但是这并不意味着停止输出，只要输出表列中还有未输出的量，将重复使用格式说明或接着按斜杠后面的格式说明组织输出.

右括号和斜杠的作用有一点不同：遇最后的括号而输出表列中已无输出的量则停止输出，或称格式控制的终止. 而斜杠只结束本行（即不能在本行上再输出其他内容），但输出仍未终止，直到无输出的量以及遇最后面的括号为止. 例如：

```
      WRITE ( * ,150) I,J
150   FORMAT (1X,2I3)
```

输出结果占一行. 而若将格式语句改为：

```
150     FORMAT (1X,2I3/)
```

在打印完 I 和 J 的值后，遇到斜杠，本行结束，但输出并未停止，再打印一个“空行”，遇右括号才结束输出. 一共打印出两个记录(两行)，其中一个是空行.

(8) FORMAT 语句可以与 WRITE 语句相邻，也可以放在程序中任一位置(但应在 END 之前，PROGRAM 语句之后，见附录Ⅳ). 一般将 FORMAT 语句集中放在程序最前或最后面，与执行语句分开. 本书为照顾初学者看程序方便，一般把 FORMAT 语句与 WRITE 语句(或 READ 语句)放在一起. 为了与执行语句相区别，一般给 FORMAT 语句取大标号(如大于 100 或某一值)，执行语句的标号取小值.

*4.4.12 不用 FORMAT 语句的格式输出

为了简化，可以直接在 WRITE 语句中指定格式说明，而不另写 FORMAT 语句. 只需在 WRITE 语句中原来写标号的位置上放入一个用撇号括起来的格式说明即可. 例如：

```
        WRITE ( *,100) X,Y,Z,I
100     FORMAT (1X,F6.2,2X,F6.2,2X,F6.2,2X,I3)
```

可写成：

```
        WRITE ( *,'(1X,F6.2,2X,F6.2,2X,F6.2,2X,I3)') X,Y,Z,I
```

可以看出：FORMAT 语句中带下划线的部分和 WRITE 语句中带下划线的部分是完全相同的. 换句话说，将 FORMAT 语句中的格式说明，用撇号括起来放入 WRITE 语句中指定的位置上即可. 注意不要漏掉一对括号. 如下面的写法是错的.

```
        WRITE ( *,'1X,F6.2,2X,F6.2,2X,F6.2,2X,I3') X,Y,Z,I
```

如果格式说明比较简单而且只为一个 WRITE 语句所引用，则直接在 WRITE 语句中写格式说明比较方便，这样在看程序时不必来回寻找 FORMAT 语句. 但是，如果格式说明比较长，或者一个格式说明需被多个 WRITE 语句所引用，则用 FORMAT 语句为好. 在 WRITE 语句中直接指定格式说明是 FORTRAN 77 新增加的功能，在 FORTRAN 66 中不能使用，有的微机上使用的 FORTRAN 77 子集也不允许这样用. 对初学者来说，建议用 FORMAT 语句.

可以看出，WRITE 语句的一般形式为：

WRITE(输出设备号，格式说明)输出表列

其中输出设备号如指定为“ * ”，代表系统事先指定的设备(一般为显示器或打印机).“格式说明”可以是一个 FORMAT 语句的标号，也可以是一个字符常数(用撇号包括起来的格式说明字符串)，甚至可以是字符表达式或字符数组(有关字符表达式和字符数组的概念见第八章).

*4.4.13 用 PRINT 语句实现格式输出

PRINT 语句不仅能用于表控输出，而且也可用于格式输出. 例如：

```
        PRINT *,A,B,C,I
```

表示表控输出. 将“ * ”改成标号，就可实现格式输出：

```
        PRINT 100,A,B,C,I
          |_____________|
          ↓
100     FORMAT(1X,3F10.2,I3)
```

注意：PRINT 后面第一项“100”，并不是要打印出常数“100”，而是“按照标号为 100 的 FORMAT 语句提供的格式说明进行输出”. 其他的概念均与前面讲述的相同.

也可以直接在 PRINT 语句中指定格式说明. 如：

```
PRINT '(1X,3F6.2)',A,B,C
       格式说明
```

初学者在写这种形式的语句时常常会弄错，而且有的微机系统上不能用这种形式的 PRINT 语句，因此，建议初学时尽量不用它.

在学习格式输出后再回过头看一下表控输出. 可以看出表控输出实际上是隐含某一格式说明的输出. 例如在本书中举的表控输出的打印结果，对整型数据隐含用 I13 编辑符，对实型数据用 F18.7 或 E15.6E3. 不同的计算机系统隐含的格式说明不同（有的用 I16，F16.7，E15.6，等等）.

§4.5 格 式 输 入

在熟悉了格式输出之后，再学习格式输入就不困难了.

4.5.1 格式输入的一般形式

格式输入指的是按指定的格式输入数据.

格式输入语句的一般形式为：

READ（输入设备号，格式说明）输入表列

与格式输出相仿，“格式说明”可以是一个标号，或者是一个以撇号括起来的格式说明字符串. 如：

```
        READ(5,200) IA,IB
          |_____________|
          ↓
200     FORMAT(I4,I3)
```

其中“200”是 FORMAT 语句的标号. 它的含义是：按标号为 200 的 FORMAT 语句所提供格式说明的要求输入数据. 也可以将上面两语句合并成以下语句：

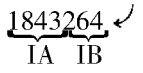
```
READ(5,'(I4,I3)')IA,IB
```

许多微机系统指定以“5”作为键盘输入的设备号. 如果写成：

```
READ(*,200) IA,IB
```

其中“ * ”代表“系统事先指定的输入设备”. 一般微机系统上指定为显示器键盘，在一些大、中型计算机系统上，指定为“读卡机”（或“软磁盘输入机”）.

对上面的格式输入语句，可以从键盘输入：



如果用卡片输入，可以在卡片的第 1 至 7 列穿七个字符，将卡片输入即可. 执行完毕后，第 1 到第 4 列的内容 1843 被读到变量 IA 中，第 5 至 7 列内容被读到变量 IB 中.

上一节介绍的 I，F，E，X，/编辑符和重复系数都可用于输入的格式说明中. 其含义基本相同. 关于它们的使用见下一小节（4.5.2 节）.

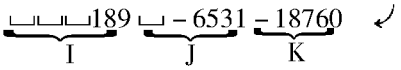
4.5.2 格式输入的规则

1. 在输入的格式说明中无需在第一项指定“1X”作“纵向走纸控制符”. 因为“走纸”显然只适用于打印输出，而输入不需要也不可能有“走纸控制”，因此，不应加“1X”. 请注意上面的 FORMAT 语句.

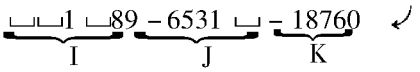
2. I 型编辑符用于整型数据的输入. 输入的两数据之间不必用逗号或空格分隔. 系统会按字段编辑描述符截取指定的列数，将其中的数据输入相应的变量中. 指定的字段宽度包括符号在内. 输入的空格按零处理. 如：

```
      READ(*,200) I,J,K
200   FORMAT(3I6)
```

如果想使 I = 189，J = −6531，K = −18760，应从键盘输入：



如果输入：



则 I = 1089，J = −65310，K = −18760.

下表是一些输入实例.

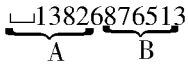
键盘输入的数据	编辑描述符	计算机内的值	说　　明
+10	I3	10	
␣−123	I5	−123	
12␣6␣	I5	12060	空格按零处理
−1346	I4	−134	只取 4 列，第 5 列无效

3. 实型数据的输入，一般用 F 型编辑描述符. 按字符宽度输入数据（数据应在字段范围右端），输入的数据可以采取两种形式：

（1）不带小数点，由计算机按指定位置自动加上小数点. 如：

```
      READ(*,300) A,B
300   FORMAT(2F6.2)
```

如果输入数据为：



系统将前 1 至 6 列中的数字 13826 作为送给变量 A 的数据，由于 F6.2 规定小数点后有二位数字，因此系统在最后二位数字之前自动加上一个小数点，成了 138.26，然后送到 A 中. 同理，送到 B 中的值为 8765.13.

（2）输入的数据本身带小数点，如果编辑描述符中规定的小数点位置和数据本身的小数点位置有矛盾，按“自带小数点优先”的原则. 如果在执行上面的读语句时输入的数据为：

$$\underbrace{13.826}_{A}\underbrace{0.8765}_{B}$$

第一个数据“自带”的小数点在最后三个数字之前，第二个数据“自带”的小数点在最后四位数之前，而 FORMAT 语句中用 F6.2，指定小数点后有二位数字. 二者矛盾. 按上面原则，将 13.826 送给 A，0.8765 送给 B.

自带小数点的输入比较直观，不易出错. 但应注意数据必须在字段范围内. 如果上述输入改为：

$$\underbrace{\text{␣}13.82}_{A}\underbrace{60.8765}_{B}$$

在第 1 列多打入一个空格，则 1 至 6 列为␣13.82（小数点也占一列），而 B 的值变成 60.876 了.

由于在“自带小数点”的数据输入时，编辑描述符 Fw.d 的 d 实际上不起作用，只有 w（宽度）起作用，因此可以将 d 定为 0. 如：

```
200        FORMAT（2F6.0）
```

效果一样.

4. 实数输入时，可以任意选用 F 型或 E 型编辑符，二者作用相同. 如：

```
           READ（*,200）A,B,C
200        FORMAT（F10.2,E10.3,F10.2）
```

输入时任意用小数形式的实数或指数形式的实数. E10.3 的作用与 F10.3 相同. 如果输入以下数据：

$$\underbrace{\text{␣␣␣}7765843}_{10\text{ 列}}\underbrace{\text{␣␣}1.18E-02}_{10\text{ 列}}\underbrace{\text{␣}0.238E+03}_{10\text{ 列}}$$

执行后将 77658.43 送给 A，1.18×10^{-2} 送给 B，0.238×10^{3} 送给 C. 可以看到，在输入时，F 和 E 型编辑符可以互换，写 F10.3 和 E10.3 都可以. 为方便起见，建议都用 F 编辑符.

下面是实型数据输入的情况.

键盘输入的数据	编辑描述符	输入后的内部值	说　　明
␣␣187326	F8.3	187.326	在最后三位数字前加小数点
48.65␣␣␣	F8.6	48.65	自带小数点优先
␣-1.8E+03	F9.2	-1.8×10^{3}	
␣85.34␣␣	E8.2	85.34	E 和 F 编辑符作用相同
1268E+02	F8.1	126.8×10^{2}	如果指数前面的数值部分不带小数点，则根据 F8.1 的规定，对其加上小数点，得 126.8E+02

␣21.5E ␣10 ␣␣	F11.3	数太大，溢出	由于 E ␣10 后面有两个空格，按零处理，成了 21.5E ␣1000，即 21.5×10^{1000}，溢出
␣␣8765 ␣31 ␣46	E11.2	8765031.04	空格作零处理，只截取 11 列，第 12 列的数无效

5. 在输入时，格式说明中的 X 编辑符的作用是“跳过若干列”. 如：

```
      READ(*,120) I,A
120   FORMAT(I3,2X,F6.2)
```

输入数据为：

146 ␣␣ 13.463 ↵

I A

即，按“I3”输入数据 146 给变量 I 后，跳过两列不读，再将后 6 列的数据 13.463 送给变量 A. 这样的好处是：有意识地使输入的两个数据间空两个空格，避免相邻的两个数据界限不清而易出错. 如果将 FORMAT 语句改为：

```
120   FORMAT(1X,I3,F6.2)
```

而输入的数据为：

123247657 ␣ ↵

则第 1 列的数字“1”被跳过不读，变成下面所表示的那样：

1 23247657 ␣

不读 I A

I 的值变成 232，A 的值为 4765.70.

6. 如果输入时，格式说明已用完而还有未被输入的变量，则重复使用该格式说明.

```
      READ(*,200) T,U,V
200   FORMAT(F6.2)
```

应从键盘输入三行信息，如：

␣11.22 ↵

86.73 ↵

␣␣12.5 ↵

如果只送入一行信息：

␣11.22 ␣86.73 ␣␣12.5 ↵

则只读入前 6 列的信息“ ␣11.22”，送给变量 T，此时格式说明已用完，遇右括号，结束本记录的输入（即不再从本行中取信息），由于还有变量 U，V 未被输入数据，故重复使用格式说明，从下一个记录中读入数据.

7. 斜杠的作用是结束本记录的输入，并接着输入下一个记录，直到遇格式说明的右括号并且已无输入的变量为止.

```
      READ(*,100) A,B
100   FORMAT(F6.2/)
```

读入 A 后遇“/”，另读新记录. 而“/”后面相当于一个空格式说明(　　)，空读一个记录，然后再重

复使用 FORMAT 语句的格式说明，读入 B，再读一个空记录. 因此，应该准备 4 个记录，其中第 1，3 两记录的数据被读入 A，B，第 2，4 两记录被跳过（即使有数据也不读入）.

8. 读语句和写语句可以共用一个 FORMAT 语句，如：

```
      READ(*,100) I
      WRITE(*,100) I
100   FORMAT(I5)
```

如果输入数据为：

47653↵

则输出为：

7653 （第一个字符被“吃掉”）

格式输入容易弄错，而且错了还不易被发现（因为输入到内存后的情况你并不知道）. 对以上规则不必死记，多上机练习几次就可清楚.

4.5.3 READ 语句的其他形式

如果用“*”作为“输入设备号”，则 READ 语句还可简化. 如：

```
READ(*,100) A,J,K
```

可以将第一个“*”号省略，括弧也可不写，而写成如下形式：

```
READ 100,A,J,K
```

但这种形式容易使人迷惑，建议少用.

```
READ(*,*) A,J,K
```

用“*”作“格式说明”，表示为“表控格式”. 它也可简化为：

```
READ *,A,J,K
```

注意省略的是第一个“*”号，从隐含的输入设备读数据. 保留的“*”是指“表控格式”. 这种形式已在本章 §4.2 中叙述过了.

§4.6 程序举例

介绍两个用输入输出语句的简单程序.

例 1 有两个直角坐标系 XOY 和 X′O′Y′. O′在 XOY 坐标系中的坐标为 (x_0, y_0)，并且 X′O′Y′在 XOY 内按反时针方向旋转 α 度.（见图 4.1）. 今有一点 P 在 XOY 坐标系中的坐标为 (x,y)，问在 X′O′Y′坐标系中其坐标是什么值？

本例要求将 P(x,y)转换成 P(x′,y′).

$$x' = (x - x_0)\cos\alpha + (y - y_0)\sin\alpha$$

$$y' = (y - y_0)\cos\alpha - (x - x_0)\sin\alpha$$

按此公式写出程序如下：

```
      READ(*,100) X,Y
      READ(*,100) X0,Y0
```

```
      READ ( * ,150) ALFA
      ALFA = ALFA * 3.1415926/180.0
      F = X - X0
      G = Y - Y0
      WRITE ( * ,200) F * COS( ALFA) + G * SIN( ALFA)
      WRITE ( * ,220) G * COS( ALFA) - F * SIN( ALFA)
100   FORMAT (2F6.2)
150   FORMAT (F6.2)
200   FORMAT (1X,'X' = ',F10.5)
220   FORMAT (1X,'Y' = ',F10.5)
      END
```

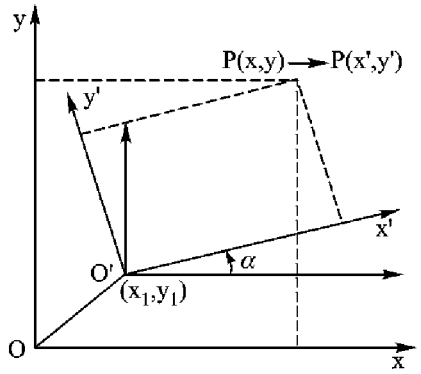


图 4.1

用第一个 READ 语句输入 P 点在 XOY 系中的坐标 P(x,y). 第二个 READ 语句输入 O′在 XOY 系中的坐标 O′(x_0,y_0). 第三个 READ 语句输入 α 角(单位为度). 然后将 ALFA(代表 α)化成弧度. 设 F 和 G 两个中间变量分别代表($x-x_0$)和($y-y_0$),以减少重复计算次数. 可以看到用 WRITE 语句可以输出表达式的值.

在本例中将 FORMAT 语句集中放在程序最后,这是结构化程序的习惯,这样可以集中精力顺序地阅读执行语句. 尤其当一个程序中 FORMAT 语句比较多时,这样做的优点更为明显. 因为 FORMAT 语句只影响输出的格式而不影响程序的结构和流程,在设计和分析程序时,可以放在稍后去处理它. 请注意 200 和 220 语句中 X 和 Y 后面是两个连续的撇号,它表示在输出时在 X 和 Y 之后要打印出一个撇号. 程序运行情况如下:

```
␣␣6.50 ␣␣6.50 ↵
␣␣6.00 ␣␣8.00 ↵
␣30.00 ↵
X' = -.31699
Y' = -1.54904
```

另一次运行记录为:

```
␣␣5.00 ␣␣5.00 ↵
␣␣3.00 ␣␣2.00 ↵
␣20.00 ↵
X' = 2.90545
Y' = 2.13504
```

例 2 借款额的计算. 假如你向银行借款,你应当先估计一下你每年能还多少钱,才能确定你最多能借多少钱. 譬如,你每月可以还 20 元,一年还 240 元,准备分 5 年还清,年利率为 12%,经过计算你可以从银行一次最多借款 865 元. 计算公式如下:

$$D = \frac{a(I^n - 1)}{(I-1)\cdot I^n}$$

式中 I = 1 + r,r 为年利率.

```
      INTEGER D
      WRITE ( * ,100)
      READ ( * ,120) N,P,R
      T =(1 + R) * * N
      A = 12 * P
      D =(T - 1) * A/(T * R)
      WRITE ( * ,200) N,P,R,D
100   FORMAT ('INPUT THE YEARS OF INSTALLMENT',
     & ' THE INSTALLMENT AND INTEREST')
120   FORMAT (I3,F6.2,F6.2)
200   FORMAT (' YEARS OF INSTALLMENT:',I3/
     &        ' MONTHLY SOUENCY:',F6.2/
     &        ' INTEREST:',F6.2/'LOAN LIMITS:',I6)
      END
```

第一个 WRITE 语句的作用是打印出一行“提示”，提醒程序员输入年数 N、每月偿还款 P 和年利率 R 的值. 然后按标号为 120 的 FORMAT 语句规定的格式输入数据.

程序中的 T 代表$(1 + r)^n$. 由于 R 代表年利率 r. 而 r = I - 1，因此，第 6 行求出的 D 等于$\frac{a(I^n - 1)}{(I - 1) \cdot I^n}$. 使 D 为整型变量，求出的金额以元为单位（不计角、分）.

运行结果如下：

```
INPUT THE YEARS OF INSTALLMENT THE INSTALLMENT AND INTEREST
005 ␣20.0 ␣␣0.12 ↵      （输入 N,P,R 的值 5,20,0.12）
YEARS OF INSTALLMENT:5
MONTHLY SOUENCY:20.00
INTEREST:  .12
LOAN LIMITS:     865
```

习　题

1. 有一表控输入语句

```
READ * ,IA,IB,IC
```

如果输入以下记录，问 IA,IB,IC 的值各为多少？

（1）87,65, - 173

（2）␣87 ␣65, - 1 ␣73

（3）87

␣65 ␣73 64　　（两个记录）

（4）␣87,65,73,84

2. 有以下两个 READ 语句：

READ ＊,X,Y,Z,A

READ ＊,C,F,G,H,Z

输入数据为：

16.8␣21.78

67.31,21.3␣6,␣560.3␣72

问各变量的值是多少？

3. 写出用下列编辑描述符输出时的打印结果.

机内数值	编辑符	输出形式
234	I6	
－8760	I8	
－6721	I4	
84	I6.4	
212.6364	F8.2	
1.321	F10.8	
2×10^5	F15.3	
621.4	F6.3	
6.52	E10.2	
-1.5×10^2	E8.1	
12.46	E14.2E3	

4. 写出用下面编辑符输入后，计算机内变量的值.

卡片上数据	编辑符	机内数值
124␣␣	I5	
1834	I3	
－234␣	I4	
246327	F6.2	
－184768	F7.3	
␣␣␣12.7803	F10.3	
␣834.263	E8.0	
1234E␣02	E8.1	
1.34E␣12	F8.2	
－346.7E＋03␣	F11.2	

5. 有下列的写语句：

WRITE（＊,100）A,B,I,J,X,Y,M,N

若 A＝1.5,B＝3.6,I＝6,J＝－40,X＝7.267,Y＝－8760.36,M＝－34,N＝678.对下列不同形式的 FORMAT 语句，分别写出打印结果.

（1）100　FORMAT（1X,F6.2,F7.3,I4,I6,F8.2,F7.1,I5,I4）

（2）100　FORMAT（1X,2F6.3,2I4,2F10.2,2I7）

（3）100　FORMAT（1X,2（2F6.3,2I5））

（4）100　FORMAT（1X,2F6.2,I7,I8）

（5）100　FORMAT（1X,2F6.2,2X,2I6,2X,2F10.1,2X,2I7）

（6）100　FORMAT（1X,'A,B,I,J,X,Y,M,N=',2(2F6.2,2I6)）

（7）100　FORMAT（F6.2,F6.2,2I4,2X,2F7.2,2I8）

（8）100　FORMAT（1X,2F6.2/1X,2I6/1X,2F7.2/1X,2I7）

（9）100　FORMAT（'A,B,I,J,X,Y,M,N='/1X,2(2F7.1,2I5)）

（10）100　FORMAT（3H ␣AB,2F6.2,2X,I6,I5）

6. 有一读语句：

READ（*,100）A,B,J,K

问对如下的 FORMAT 语句，应如何输入数据（注明列数）. 设 A 的值为 6.3，B 为 -76.2，J 为 17，K 为 -12.

（1）100　FORMAT（F6.2,F5.2,I2,I4）

（2）100　FORMAT（2F10.3,2I6）

（3）100　FORMAT（1X,F6.2,2X,F6.3,2X,I5,2X,I4）

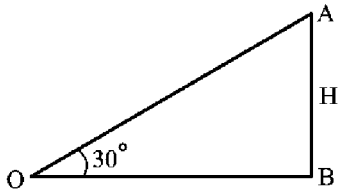
（4）100　FORMAT（F6.2,F7.2/I4,I6）

（5）100　FORMAT（E8.2,E6.3,2I4）

（6）100　FORMAT（F6.1,F7.2,I3/I4）

7. 已知梯形的上底 a = 112 厘米，下底 b = 240 厘米，高 h = 150 厘米，求梯形面积（以米^2 为单位表示，取一位小数，对第二位小数四舍五入）. ① a,b,h 用格式输入和格式输出. ② 用表控输入和表控输出. 对①，②请分别编写程序.

8. 雷达在仰角 $\alpha = 30°$ 时发出电波碰着飞机（A 点），经过 1/3000 秒收到来自飞机的回波，电波速度为 3×10^8 米/秒，问飞机的高度（见右图）. 请编写程序，用表控格式输入仰角 α，用格式输出 H. 在输出时加上适当的文字说明（用英语或拼音）.



（题 8）

第五章　逻辑运算和选择结构

在前两章中介绍的程序中，只用到了顺序结构，即语句执行的顺序和语句在程序中的物理顺序（即先后位置）是一致的. 由于程序中某些结构的要求，有时需要有意识地改变语句执行的顺序，或根据给定条件是否满足决定执行某一个“分支”.

§5.1　无条件转移语句（GO TO 语句）

它的形式为：

GO TO　　n

其中 n 为本程序单位中一个执行语句的标号. 在执行 GOTO 语句时，流程转到标号为 n 的语句处并接着执行. n 是一个无符号的数值常数，范围是 1 ~ 99999.

可以利用 GOTO 语句使程序实现循环. 有了 GOTO 语句可以使程序更加方便灵活. 但必须着重指出：不要滥用 GOTO 语句，否则会破坏结构化原则. 只应该在符合结构化原则的前提下恰当地使用 GOTO 语句.

例　从终端键盘不断输入一批正数，将它们累加并打印.

```
        T=0
1       READ(*,*)X
        U=T
        T=T+X
        WRITE(*,100)U,X,T
100     FORMAT(1X,F6.2,'+',F6.2,'=',F6.2)
        GOTO 1
        END
```

运行情况如下：

```
12.4 ↵  （输入 12.4）
      .00 + 12.40 = 12.40
23.6 ↵
    12.40 + 23.60 = 36.00
34.0 ↵
    36.00 + 34.00 = 70.00
```

请注意 FORMAT 语句中的格式说明及其在输出中的作用.

这个程序能正确地运行，但它永远不会停止，因为永远不会执行 END 语句，需要程序员人工干预使之强制停止. 因此它不是一个好的程序. 我们应当使用“条件控制”的方法使之自动停止，

即满足给定条件时停止执行. 下一节介绍的逻辑 IF 语句可以解决这个问题.

§5.2 逻辑 IF 语句

这是一种最简单的选择结构,用一个语句构成一个选择结构. 例如:

```
IF(X.GE.0) WRITE(*,*) X
```

它的作用是,当 X 的值大于或等于 0 时,打印出 x 的值. 它的流程见图 5.1 和图 5.2.

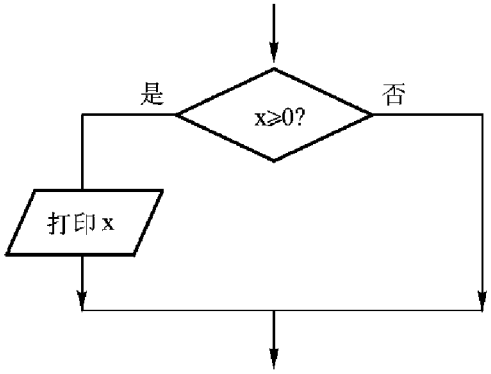


图 5.1

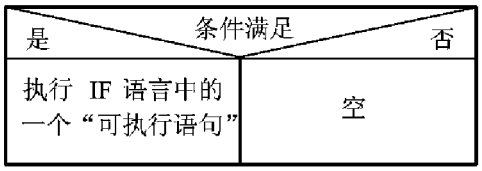


图 5.2

如果 X < 0,则不执行本语句中 WRITE 语句而执行 IF 语句的下一个语句.

逻辑 IF 语句的一般形式为:

IF (〈条件〉) 可执行语句

即给定的条件满足,就执行后面的“可执行语句”.

注意:只能跟一个可执行语句,而不能跟若干个可执行语句. 例如下面的写法是错误的:

```
IF (A.GT.B) T = A A = B B = T
```

原意是:如果 A > B,则使 A,B 互换. 但在编译时将显示出“语法错误”的信息.

下面先举两个例子说明逻辑 IF 的应用.

例 **1** 某货物单价 850 元,买 100 个以上(包括 100 在内)按九五折优惠价. 输入购买个数,求总货款.

```
PRICE = 850.0
READ(*,*) N
IF(N.GE.100)      PRICE = PRICE * 0.95
A MOUNT = N * PRICE
WRITE(*,*) 'N = ',N,'      AMOUNT = ',AMOUNT
END
```

流程图和结构流程图见图 5.3 和图 5.4.

例 **2** 输入 A,B 两个数,按大小顺序打印出来.

```
READ(*,*) A,B
IF(A.GE.B)  GOTO  10
T = A
A = B
```

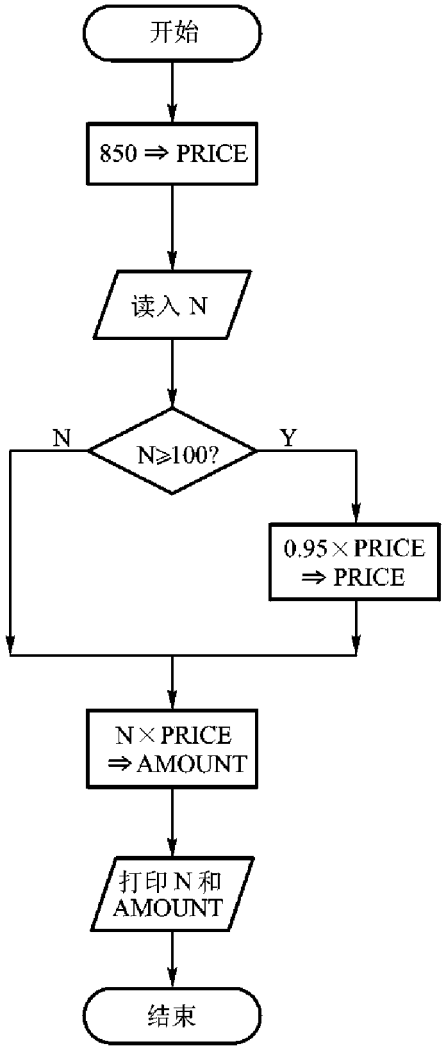


图 5.3

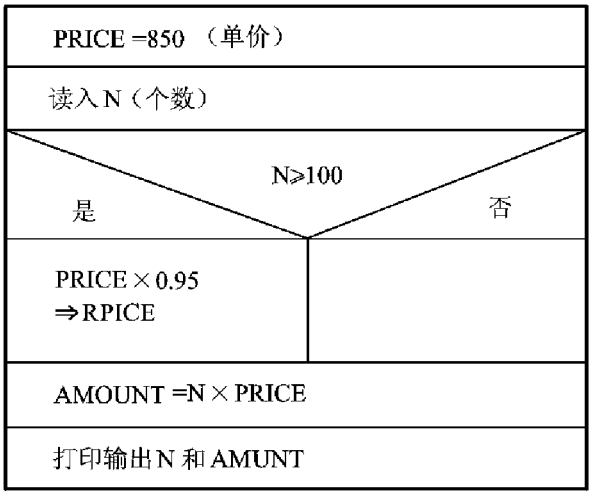


图 5.4

```
      B = T
10    WRITE ( * ,100) A,B
100   FORMAT (1X,'MAX = ',F10.2,' MIN = ',F10.2)
      END
```

请自己写出流程图.

§5.3 关系表达式

上述逻辑 IF 语句中的“条件”,最简单的形式是一个“关系表达式”. 例如,“X. GE. 0”或“A. GT. B”就是关系表达式. 它将两个量进行比较. 两个算术表达式用一个关系运算符连接起来的式子就是算术关系表达式. 例如:

```
(A + B). GT. (C + D)
SIN(X). EQ. -3
B * B - 4 * A * C. GE. 0
```

都是合法的算术关系表达式形式. 它的一般形式为:

〈算术表达式〉〈关系运算符〉〈算术表达式〉

FORTRAN 77 提供六种关系运算符(又称关系比较符).它们是:

.LT.　　(小于)

.LE.　　(小于或等于)

.EQ.　　(等于)

.NE.　　(不等于)

.GT.　　(大于)

.GE.　　(大于或等于)

注意:

1. 关系运算符两侧必须有句点“.”作为分界符.如果将(A.GT.B)写成(AGTB),系统就会将 AGTB 当作一个变量名了.

2. FORTRAN 77 允许将不同类型的算术量进行比较,在执行时会自动转换成同一类型(转换的规律与混合运算时的规律相同,即低级的向高级的转换).如:

A.EQ.(5-2)

系统会先将 5-2 的结果 3 化成实数 3.0,然后与实型变量 A 比较.

3. 关系表达式的运算次序是,先计算算术表达式的值,然后再进行比较.换句话说,先执行算术运算符的操作,再进行关系运算符的操作.因此上面的关系表达式可写成:

A.EQ.5-2

对“5-2”加括弧或不加括弧作用相同.因为先执行算术操作.

归纳起来,算术关系表达式执行步骤为:(1)先计算算术表达式的值;(2)将两个算术表达式转换为同一类型;(3)将两个算术表达式进行比较.

4. 关系表达式的值不是一个数值,而是一个“逻辑量”,即“真”或“假”.例如,当 A=5.5,B=3.6 时,关系表达式:

A.GT.B

的值为“真”.如果 A=0.5,则关系表达式的值为“假”.为了便于理解,可以将“真”理解为“条件满足”,将“假”理解为“条件不满足”.

5. 在用.EQ.或.NE.比较符对两个实数进行比较时,有时会出现一些小误差.例如将 10 个 0.1 累加,结果可能不等于 1,这是由于实数在内存中以二进制形式存放时出现的误差.0.1 是不可能用有限位的二进制数准确地表示的.因此,(10*0.1.EQ.1)或(1.0/3.0*3.EQ.1)的值不是“真”而是“假”.因此,应估计可能出现的误差,如果从理论上计算 A 应等于 B,不要写成:

```
IF (A.EQ.B) …
```

或

```
IF (A.NE.B) …
```

而应写成:

```
IF （ABS（A－B）.LT.1E－5） …
```

即当（A－B）的绝对值（也就是两个数之差）小于某一个很小的数时，便认为此二数相等.

```
IF （ABS（A－B）.GT. 1E－5） …
```

即当两个数的差大于某一个很小的数时，才认为此二数不相等.

§5.4 逻辑表达式

上节中举的例子里，逻辑 IF 语句中用的“条件”是简单的条件，如（A.GT.B），（T.LE.1.0）等.但在实际问题中，有时需要处理的是“复合条件”.例如：“当 A＞B 和 A＞C 时，打印 A 的值”.这里“A＞B”和“A＞C”是两个简单条件，而“A＞B 和 A＞C”是一个“复合条件”.逻辑 IF 语句可以写成：

```
IF （A.GT.B .AND. A.GT.C） WRITE（＊，＊）A
```

其中“A.GT.B .AND.A.GT.C”是用“.AND.”将两个关系式联接起来，“.AND.”是“与”的意思，表示当它两侧的关系表达式的值都为“真”时，（A.GT.B .AND.A.GT.C）的值才是“真”.或者说，当 A＞B 和 A＞C 两个简单条件都满足时，复合条件“A.GT.B.AND.A.GT.C”才满足，只要 A＞B 或 A＞C 二者之一不满足，“A.GT.B.AND.A.GT.C”就不满足.

上面的（A.GT.B.AND.A.GT.C）就是一个逻辑表达式.逻辑表达式是对逻辑量进行逻辑运算的，.AND.是一个逻辑运算符.逻辑表达式的值是一个逻辑值（“真”或“假”）.

逻辑表达式的最简单的形式为：

逻辑常数

逻辑型变量

关系表达式

5.4.1 逻辑常数

逻辑常数只有两个：

.TRUE. “真”.表示满足逻辑条件.

.FALSE. “假”.表示不满足逻辑条件.

5.4.2 逻辑变量

逻辑变量用来存放逻辑常数.因此，逻辑变量的值也只有两种可能：.TRUE.（真）或.FALSE.（假）.

逻辑变量需要用“逻辑说明语句”来说明.如：

```
LOGICAL A,B
```

A 和 B 被定义为逻辑变量.也可以用

```
IMPLICIT LOGICAL （A,B）
```

表示凡以 A 和 B 开头的变量均为逻辑变量.

可以用赋值语句对逻辑变量赋以一个逻辑常数.如：

A = . TRUE.

B = X. GT. 5.6

经赋值后 A 的值为.TRUE.（真），如果变量 X = 3.0，则逻辑变量 B 的值为. FALSE.（假）.

5.4.3 逻辑运算符

FORTRAN 77 提供了五个逻辑运算符：

. AND.　　逻辑与

. OR.　　逻辑或

. NOT.　　逻辑非

. EQV.　　逻辑等

. NEQV.　　逻辑不等

举例如表 5.1.

表 **5.1**

逻辑表达式	逻辑表达式的值
X + Y. LT. 0	当 x + y < 0 时，表达式的值为真
（A + B. EQ. C）. AND.（T. EQ. Q）	当 A + B = C 和 T = Q 同时成立时，为真
. NOT. . TRUE.	恒为假
. NOT.（X. GT. 0）	当 x > 0 时，表达式值为假
（A. GE. B）. OR.（C. GE. D）	当 A≥B 或 C≥D 之一成立时，表达式值为真
. NOT. FLAG1. AND. FLAG2	当逻辑变量 FLAG1 值为假，FLAG2 值为真时，表达式值为真
L1. EQV. L2	如果 L1 和 L2 均为相同的逻辑值（同时为真或同时为假），则为真
L1. NEQV. L2	如果 L1 和 L2 的逻辑值不相同（一为真，另一为假），则表达式值为真

表 5.2 为五种逻辑运算的真值表

表 **5.2**

A	B	. NOT. A	. NOT. B	A. AND. B	A. OR. B	A. EQV. B	A. NEQV. B
真	真	假	假	真	真	真	假
真	假	假	真	假	真	假	真
假	真	真	假	假	真	假	真
假	假	真	真	假	假	真	假

5.4.4 逻辑表达式的运算次序

在一个逻辑表达式中往往包含算术运算符、关系运算符和逻辑运算符，它们的运算优先次序如表 5.3.

表 5.3

类　别	运　算　符	优　先　级
括号	（　）	1
算术运算符	* *（乘方）	2
	*，/	3
	+，-	4
关系运算符	.GT.，.GE..LT.， .LE.，.EQ.，.NE.	5
逻辑运算符	.NOT.	6
	.AND.	7
	.OR.	8
	.EQV.，.NEQV.	9

例如：若 A = 1.5，B = 2.0，C = 1.2，D = 7.5，X = 3.0，Y = 5.0，L1 为真，有逻辑表达式：

A.GT.3.6 * B .AND. X.EQ.Y. OR.L1.AND. .NOT.(3.6 - C) * 2.GE.D/2.5

① 2.4

② 7.2　③ F　　③ F　　② 4.8　② 3.0　③ T　④ F　⑤ F　⑤ F　⑥ F

运算次序为：

① 先算出括弧内的值，(3.6 - C)的值为 2.4.

② 算出算术表达式 3.6 * B，(3.6 - C) * 2 和 D/2.5 的值，它们分别为 7.2，4.8 和 3.0.

③ 算出关系表达式 A.GT.3.6 * B，X.EQ.Y 和(3.6 - C) * 2.GE.D/2.5 的值，它们分别是 F(假)，F(假)和 T(真).

④ 进行.NOT.运算，.NOT.(3.6 - C) * 2.GE.D/2.5 的值为假.

⑤ 进行.AND.运算，.OR.前面和后面的两个逻辑表达式的值均为假.

⑥ 最后进行.OR.运算，得到整个逻辑表达式的值为假.

在学习了逻辑表达式的概念之后，可以知道逻辑 IF 语句中的“条件”就是逻辑表达式. 逻辑 IF 语言的一般形式为：

IF（〈逻辑表达式〉）　可执行语句

§5.5　逻辑数据的输入输出

逻辑数据的输入和输出可以用表控方式，也可以用格式的输入输出.

5.5.1 用表控格式输入输出逻辑数据

```
READ( *, * ) L1, L2
WRITE( *, * ) L1, L2
```

假设 L1, L2 已定义为逻辑型数据，如果想使 L1 为真，L2 为假. 只要输入：

```
.TRUE. , . FALSE.
```

或

```
T, F
```

即可.

执行 WRITE 语句时打印出 T(代表.TRUE.)和 F(代表.FALSE.)：

```
␣␣␣T ␣␣␣F
```

每一字段宽由具体的计算机系统规定.

5.5.2 用格式输入输出逻辑数据

用 L 编辑描述符规定逻辑数据输入输出的格式.

```
      READ( *,100) L1, L2
100   FORMAT(2L2)
      WRITE( *,200) L1, L2
200   FORMAT(1X,L4,2X,L4)
```

L2 表示逻辑数据的输入输出，字段宽为 2. 执行时输入：

```
␣T␣F
2 2
```

将.TRUE. 输入给 L1，将.FALSE. 输入给 L2. 输出为：

```
␣␣␣T␣␣␣␣␣F
L1      L2
```

§5.6 块 IF

逻辑 IF 语句虽然简单方便，但它的缺点是：

(1) 在两个分支中有一个是“空块”. 也就是，当条件不满足时不执行任何操作(见图 4.1 和图 4.2).

(2) 当条件满足时，只能执行一个可执行语句，而不能执行一组可执行语句.

因此，当需要处理的两个分支都包含若干个语句时，用逻辑 IF 语句就有困难了. 而用块 IF 能够较好地解决这个问题. 我们先来看一个例子.

例 1 有一函数(见图 5.5)，要求从键盘输入一个 x 值($x \geq -4$)，并打印出相应的函数值.

由图可建立数学模型：

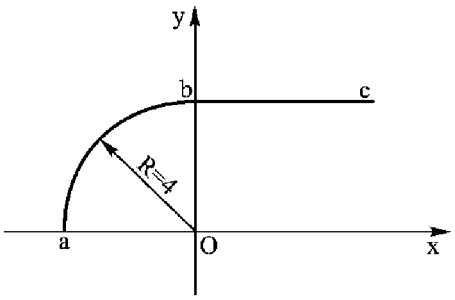


图 5.5

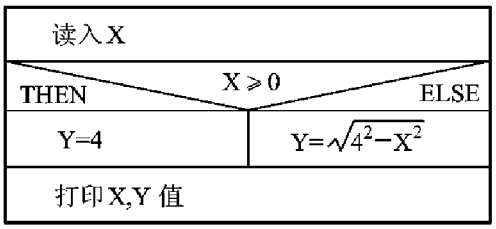


图 5.6

$$y=\begin{cases}\sqrt{4^2-x^2} & (-4\leqslant x<0)\\ 4 & (x\geqslant 0)\end{cases}$$

先画出结构流程图(图 5.6),由于已知 x≥ -4,故读入 x 后只需判别 x 是否≥0 即可.

程序如下:

```
READ(*,*) X
IF(X.GE.0)      THEN                    (块 IF 语句)
   Y=4.0                                (IF 块)
ELSE                                    (ELSE 语句)
   Y=SQRT(4.0**2-X**2)                  (ELSE 块)
END IF                                  (END IF 块)
WRITE(*,*)'X=',X,'   Y=',Y
END
```

IF 块

运行记录为:

```
-2 ↵
X=              -2.0000000 Y=           3.4641020
```

再运行一次:

```
5 ↵
X=               5.0000000 Y=           4.0000000
```

从“块 IF 语句”开始到 END IF 语句结束,是一个“块 IF”.之所以称为“块 IF”,是因为它是由若干个语句(作为一组或一块)组成一个选择结构的.这是和逻辑 IF 语句不同之处(逻辑 IF 语句只由一个语句组成一个选择结构).因此,逻辑 IF 语句又称为“行 IF 语句”,以区别于“块 IF 语句”(“逻辑 IF 语句”这个名字是在 FORTRAN 66 标准中为了与“算术 IF 语句”相区别而命名的).

5.6.1 块 IF 的组成

一个块 IF 一般可写成以下形式:

```
IF(逻辑表达式)  THEN          (块 IF 语句)
      块 1                    (IF 块)
ELSE                          (ELSE 语句)
```

块 2　　（ELSE 块）

END IF　　（END IF 语句）

其相应的流程图见图 5.7.

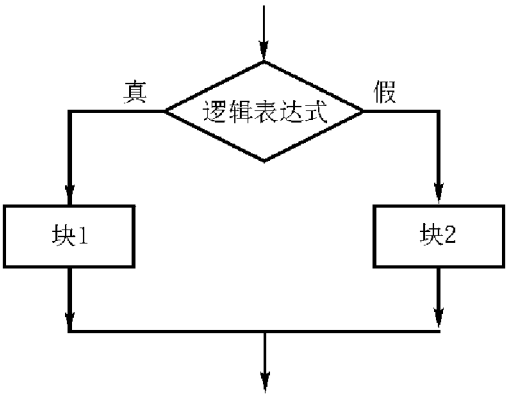


图 5.7

其中 ELSE 语句和 ELSE 块可以没有. 即块 2 可以是“空块”，如：

```
IF （逻辑表达式） THEN
        IF 块
END IF
```

有的块 IF 中还可以包括 ELSE IF 语句和 ELSE IF 块. 见 §5.6.4.

块 IF 语句、ELSE 语句、END IF 语句都是可执行语句.

5.6.2 块 IF 的执行过程

1. 先执行块 IF 语句，判断逻辑表达式的值，如果为真，执行 IF 块中的可执行语句. IF 块的范围由块 IF 语句下面第一个语句开始直到同一个块 IF 中的 ELSE 语句、ELSE IF 语句或 END IF 语句为止（不包括这些语句本身）.

2. 如果逻辑表达式的值为假，不执行 IF 块，而使流程转到 ELSE 语句. ELSE 虽然是执行语句，但它本身不进行任何操作，只是将流程引向 ELSE 块，执行 ELSE 块中各执行语句. 从程序中看，ELSE 语句的作用是把 IF 块和 ELSE 块分隔开来，ELSE 块的范围由 ELSE 语句之后到与之对应的 END IF 之间的全部执行语句.

3. 在执行完 IF 块或 ELSE 块中所有执行语句后，流程都转到 END IF 语句. 结束本“块 IF”的操作. END IF 的作用是确定 ELSE 块或 IF 块（当无 ELSE 语句和 ELSE 块时）的范围，为块 IF 提供一个“出口”. 执行 END IF 后应接着执行 END IF 后面的语句. 千万不要漏写 END IF 语句，否则系统就会将它后面的语句都作为 ELSE 块或 IF 块的一部分，而发生逻辑错误.

注意：

（1）一个块 IF 是一个完整的选择结构. 可以由 IF 块或 ELSE 块中用转移语句将流程转到块 IF 之外，而不允许由 IF 块和 ELSE 块外转到 IF 块或 ELSE 块内. 下面的写法是错误的：

```
      IF （I.NE.J） THEN
          I=J
10        WRITE（*，*） I
      END IF
      GOTO 10
```

虽然允许由块内转到块外，但结构化原则不提倡这样做.

（2）可以在同一个 IF 块或 ELSE 块内用转移语句从一点转移到另一点.

（3）块 IF 语句、ELSE 语句和 END IF 语句必须联合使用而不能单独使用. 一个块 IF 语句必

须对应一个 END IF 语句(或一个 ELSE 语句和 END IF 语句).

5.6.3 块 IF 的嵌套

在一个块 IF 中还可以嵌套另一个块 IF. 例如:在求一元二次方程的根时,判别式 $D = B^2 - 4AC$ 的值大于零、等于零或小于零,对应三种处理方法.

```
D = B * B - 4 * A * C
IF (ABS(D) .LT. 1E-6) THEN
    X = -B/(2 * A)
    PRINT *,'X = ',X
ELSE
    IF (D.GT.0) THEN
        X1 = (-B + SQRT(D))/(2 * A)
        X2 = (-B - SQRT(D))/(2 * A)
      PRINT *,'X1 = ',X1,'  X2 = ',X2
    ELSE
        PRINT *,'X1 = ',-B/(2 * A),'+',SQRT(-D)/(2 * A),'I'
        PRINT *,'X2 = ',-B/(2 * A),'-',SQRT(-D)/(2 * A),'I'
    END IF
END IF
```

流程图见图 5.8.

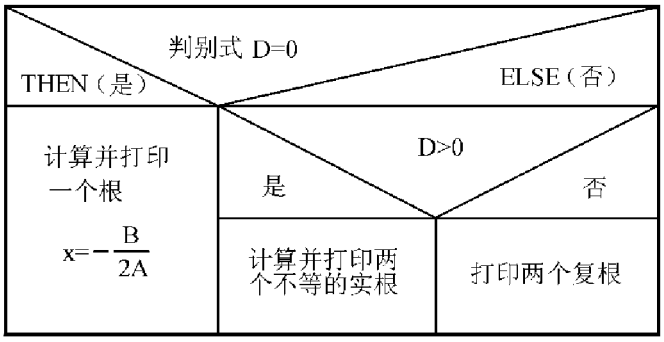


图 5.8

可以看到:要进行两层判断处理. 第一层先判别 D 是否等于零. D 是实数,前面已提到过,在用. EQ. 或. NE. 等关系比较符比较两个实数时,可能会出现误差,因此,我们用"$|D| < 10^{-6}$"这一条件来代替"D = 0"条件,这样比较稳妥些. 如果判断的结果为"真",则计算并打印一个根(即两个相等的实根). 如果为"假",还要作第二层的判断,根据 D > 0 或 D < 0 作不同的处理.

程序中有两个块 IF,已分别用线括出它们的范围. 第二个块 IF 是第一个块 IF 的一部分. 在本程序中它就是第一个(外层的)块 IF 中的 ELSE 块. 也就是说,在一个 IF 块或 ELSE 块中,还可以出现另一个块 IF.

现在程序中有两个块 IF 语句、两个 ELSE 语句和两个 END IF 语句. 必须十分清楚每一个块 IF 的范围以及每一层块 IF 所包括的块 IF 语句、ELSE 语句和 END IF 语句. 在写程序时,为了使阅读方便,往往写成"锯齿形". 即同一层的块 IF 语句、ELSE 语句和 END IF 语句写在同一列上. 内层的块 IF 向右缩进几列. 这样层次分明,不致搞混.

嵌套可以不限于两层，可以在内层的块 IF 中再出现第三层块 IF，如此一层层嵌套下去.

当嵌套层次多时，往往一时难以找出同一层的块 IF 中的各语句. 可以按以下方法确定各层块 IF：

（1）从最内层的块 IF 语句开始，向下找到离它最近的 END IF 语句，把它们用线括起来，在这两个语句之间的全部语句（包括块 IF 语句和 END IF 语句）就是同一层的块 IF.

（2）由内向外重复上述过程，直到遇到最外层的块 IF 语句和 END IF 语句为止.

5.6.4 ELSE IF 语句

当需要进行多分支处理时，固然可以用多层块 IF 嵌套来解决，但程序会比较长，不大方便. 例如，征收税款，税率与收入有关. 若规定收入 1000 元以下收 3%，1000～2000 元收 4%，2000～3000 元收 5%，3000 元以上收 6%. 当然可以写成：

```
    ⋮
IF (AMOUNT.LT.1000.0) THEN
   RATE=0.03
ELSE
   IF (AMOUNT.LT.2000.0) THEN
      RATE=0.04
   ELSE
      IF (AMOUNT.LT.3000.0) THEN
         RATE=0.05
      ELSE
         RATE=0.06
      END IF
   END IF
END IF
TAX = AMOUNT * RATE
```

这里用了三个块 IF 的嵌套. 如果税率分的档次更多，则嵌套层次就更多了. 可以用 ELSE IF 语句来处理这种多分支的问题. 上面的程序可以改写如下：

```
      ⋮
IF (AMOUNT.LT.1000) THEN                          (块 IF 语句)
      RATE=0.03                                   (IF 块)
ELSE IF (AMOUNT.LT.2000) THEN                     (ELSE IF 语句)
      RATE=0.04                                   (ELSE IF 块)
   ELSE IF (AMOUNT.LT.3000) THEN                  (ELSE IF 语句)
      RATE=0.05                                   (ELSE IF 块)
   ELSE                                           (ELSE 语句)
      RATE=0.06                                   (ELSE 块)
END IF                                            (END IF 语句)
TAX=AMOUNT*RATE
```

说明：

（1）ELSE IF 也是一个执行语句，它的作用是将 ELSE 语句和块 IF 语句的作用结合起来，表示：“否则，如果满足条件，则执行下面的 ELSE IF 块。”

（2）ELSE IF 块的范围从 ELSE IF 语句的下一个语句开始到离它最近的 ELSE 语句或 ELSE IF 语句或 END IF 语句为止.

（3）ELSE IF 语句的一般形式为：

ELSE IF（逻辑表达式） **THEN**

（4）它的执行规律是这样的：如果逻辑表达式的值为“真”，则执行其后的 ELSE IF 块. 如果为“假”，则转到紧跟在 ELSE IF 块后面的 ELSE 语句或 ELSE IF 语句或 END IF 语句. 请读者结合上面的程序来理解它.

（5）ELSE IF 语句实际上也引入了块 IF 的嵌套. 因为在“否则”之后又进行下一层的判断. 上页两个程序的作用是完全相同的. 可以用同一个流程图来表示. 见图 5.9 和图 5.10.

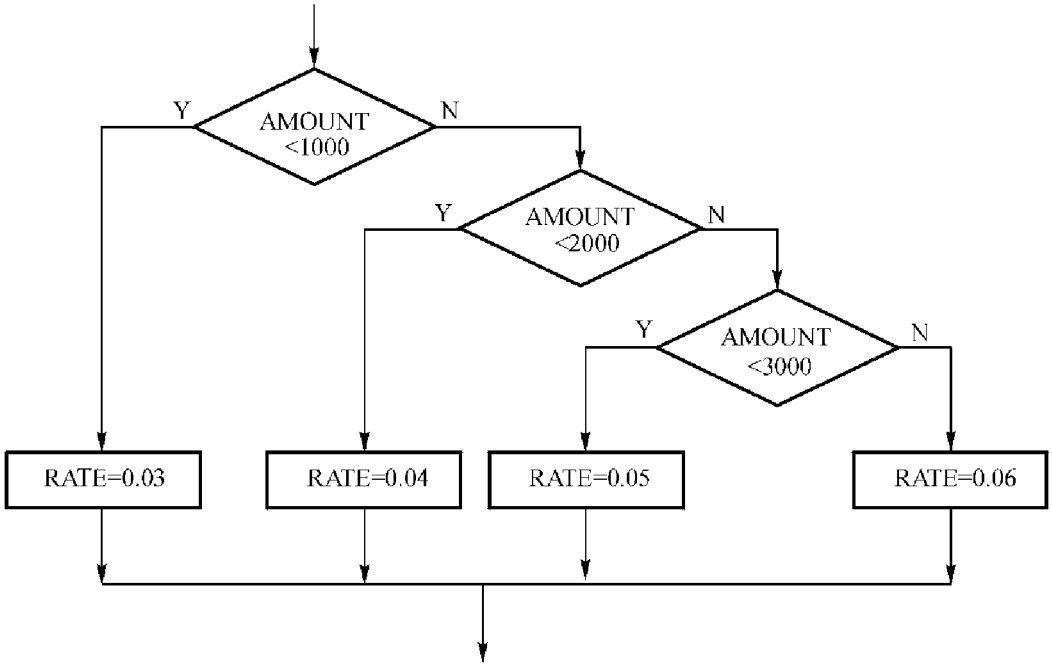


图 5.9

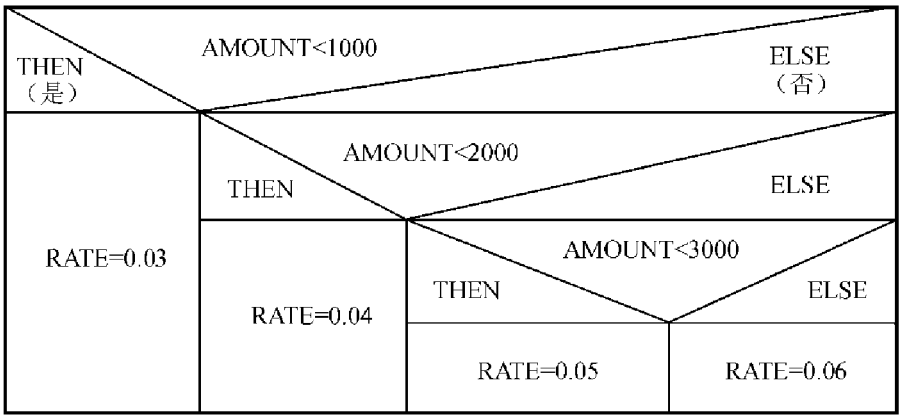


图 5.10

当分支的个数比较多时，画出以上两种形式的流程图都是不方便的. 可以将图 5.10 简化成

图 5.11 形式.

根据 AMOUNT 的值选择 RATE			
AMOUNT < 1000	1000 ≤ AMOUNT < 2000	2000 ≤ AMOUNT < 3000	AMOUNT ≥ 3000
RATE = 0.03	RATE = 0.04	RATE = 0.05	RATE = 0.06

图 5.11

用这种形式的流程图表示多分支是很方便而且一目了然的. 它的一般形式可以用图 5.12 表示.

若条件 1 满足	否则,若条件 2 满足	否则,若条件 3 满足	否 则
块 1	块 2	块 3	块 4

图 5.12

(6) ELSE IF 虽然起 ELSE 和 IF 的双重作用,但它与块 IF 语句有一点不同,它不要求有相应的 END IF 语句. 请看 94 页下部的程序中,只有一个 END IF 语句,它是和开头的块 IF 语句相对应的. 在块 IF 中有两个 ELSE IF 语句,它们并无相应的 END IF 语句. 如果写成

```
ELSE IF (AMOUNT.LT.3000) THEN
    RATE = 0.05
ELSE
    RATE = 0.06
END IF
END IF
```

反而是画蛇添足了.

(7) 在包含 ELSE IF 语句的块 IF 中,如果块 IF 语句中逻辑表达式的值为假,则流程转到 ELSE IF 语句继续执行. 这是对 §5.6.2 中第 2 点的一个补充.

ELSE IF 块的后面可以有 ELSE 语句和 ELSE 块,也可以没有,而是另一个 ELSE IF 语句或 END IF 语句.

(8) 在多分支处理中,应把出现几率较高的条件放在前面. 例如上一个例子中,当大多数的 AMOUNT 是小于 1000 时,那样写是较好的,因为在执行完 IF 块后即转到 END IF. 而如果大多数 AMOUNT 是大于 3000 时,则那样安排是不理想的,因为要经过三次判断才转到 ELSE 块,执行完 ELSE 块后才转到 END IF 语句. 当多次重复执行此块 IF 时,效率是不高的.

(9) 最后,希望读者千万不要把“块 IF”和“块 IF 语句”相混淆,前者是整个选择结构,而后者只是一个语句(块 IF 的第一个语句).

例 **2** 有一函数 x(t)随时间 t 变化的曲线如图 5.13 所示. 当输入时间 t,求相应的函数值.

由图建立数学模型:

$$x(t)=\begin{cases}\text{无定义} & (t<0)\\ x_0+\dfrac{(a-x_0)t}{t_1} & (0\leqslant t<t_1)\\ a & (t_1\leqslant t<t_2)\\ \dfrac{a(t_3-t)}{t_3-t_2} & (t_2\leqslant t<t_3)\\ 0 & (t_3\leqslant t)\end{cases}$$

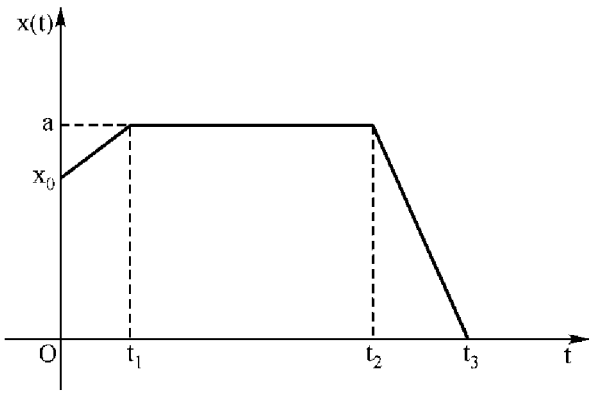


图 5.13

先画出流程图(见图 5.14).

输入 A,X0,T1,T2,T3				
输入 T				
T≥T3	T2≤T<T3	T1≤T<T2	0≤T<T1	T<0
X=0	$X=A\cdot\dfrac{T3-T}{T3-T2}$	X=A	$X=X0+\dfrac{(A-X0)T}{T1}$	无定义停止运行
打印 T 和 X 的值				

图 5.14

程序如下:

```
READ(*,*) A,X0,T1,T2,T3
READ(*,*) T
IF (T.GT. T3) THEN
    X=0
ELSE IF (T.GE.T2) THEN
    X=A*(T3-T)/(T3-T2)
    ELSE IF (T.GE. T1) THEN
        X=A
        ELSE IF (T.GE.0) THEN
            X=X0+(A-X0)*T/T1
            ELSE
                WRITE (*,*) 'NO DEFINITION'
                STOP
END IF
WRITE(*,*) 'T=',T,' X=',X
END
```

先输入 A,X0,T1,T2,T3 的值,在计算式子中它们是常数.再输入 T 的值,T 是任意的,每次可取不同的值.

运行记录如下:

```
50,40,5,20,25 ↵
```

```
3
T =          3.0000000    X =          46.0000000
```

另一次运行记录如下：

```
50,40,5,20,25  ↵
22
T =          22.0000000    X =          30.0000000
```

在本章中，我们着重介绍了用块 IF 实现选择结构. 块 IF 是 FORTRAN 77 所增加的一个重要的功能. 如果读者所使用的计算机系统没有 FORTRAN 77 的编译系统而仍用 FORTR N Ⅳ（即 FORTRAN 66 标准）的话，可以用逻辑 IF 语句来代替块 IF 来实现两个分支的选择结构.

例 **3**　输入一个数，判别它是否能被 3 整除，并打印出能被 3 整除或不能被 3 整除的信息.

程序可以写成下面的形式：

```
      READ(5,100) N
      IF (MOD(N,3)  .EQ. 0) GOTO 10
      WRITE (6,110) N
      GOTO 20
10    WRITE(6,120) N
      M = N/3
      WRITE(6,130) N,M
20    STOP
100   FORMAT(I5)
110   FORMAT(1X,I6,31H CANNOT BE DIVIDED EXACTLY BY 3)
120   FORMAT(1X,I6,28H CAN BE DIVIDED EXACTLY BY 3)
130   FORMAT(1X,I5,3H/3 = ,I5)
      END
```

运行记录如下：

```
36  ↵
  36 CAN BE DIVIDED EXACTLY BY 3
  36/3 =      12
```

它是用FORTRAN Ⅳ语句来实现结构化程序，仍可画出结构化流程图（见图5.15）. 可以

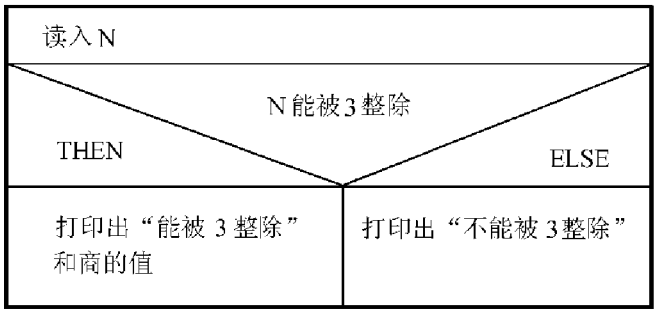


图 5.15

看到，它是用 GOTO 语句使两个分支汇合到同一个出口（STOP 语句）的. 结构化程序并不一律禁

止使用 GOTO 语句，它允许在同一个结构中使用 GOTO 语句以达到实现本结构（今为选择结构）的要求. 当然用 FORTRAN Ⅳ来实现结构化程序是不大方便的. 本例如果用块 IF，是很方便且易于阅读的. FORTRAN 66 中无表控输入输出，不能在 WRITE 语句中使用表达式，不能使用撇号编辑符和用撇号的字符常数. 本例全部按 FORTRAN 66 的规定书写的. 希望读者能通过本例举一反三，需要时能将本书中以 FORTRAN 77 书写的程序改写为 FORTRAN Ⅳ程序.

也可以直接用逻辑 IF 语句中的执行语句来实现分支的要求，本例可改写为下面形式（流程图见图 5.16）：

```
      READ(5,100) N
      IF (MOD(N,3) .EQ. 0) WRITE(6,120) N
      IF (MOD(N,3). NE. 0) WRITE(6,120) N
20    STOP
100   FORMAT(I5)
110   FORMAT(1X,I6,31H CANNOT BE DIVIDED EXACTLY BY 3)
120   FORMAT(1X,I6,28H CAN BE DIVIDED EXACTLY BY 3)
      END
```

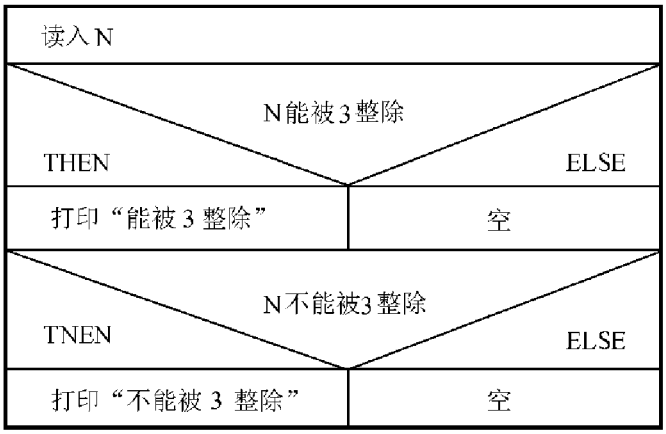


图 5.16

可以看到，由于逻辑 IF 语句中只能包括一个可执行语句，因此只能处理最简单的分支（即分支中只包括一个语句）. 请读者思考：这个程序中第三行为什么再要用 IF 语句规定条件“N 不能被 3 整除”. 如果将这两句改为：

```
      IF (MOD(N,3) .EQ.0) WRITE (6,120) N
      WRITE (6,110) N
```

会出现什么结果，请自己画出流程图分析.

习　题

1. 请写出下列逻辑表达式的值，假设 A = 2，B = 7.5，C = −3.6，逻辑变量 L1 = .TRUE.，L2 = .FALSE..

（1）A − 7.LT.B − 6.5

（2）L2

（3）C－A.EQ.B＊A－C.OR.L1

（4）.NOT.L2.OR.B－A.LE.C/2.AND.C.EQ.－3.6

（5）L1 .EQV. .NOT. L2 .AND. L1 .OR. 3＊A.EQ.4－B

2. 输入一个数，判断它能否被3或5整除，如能整除则将此数打印出来，否则不打印. 编出程序.

3. 输入一个班的平均成绩G. 如果：G≥90分，打印“VERY GOOD”；80≤G＜90，打印“GOOD”；60≤G＜80，打印“PASS”；G＜60，打印“FAIL”.

4. 有一函数（如图5.17）：

$$y=\begin{cases}\frac{40}{15}x+10 & (0\leqslant x<15)\\ 50 & (15\leqslant x<30)\\ 50-\frac{10}{15}(x-30) & (30\leqslant x<45)\\ 40+\frac{20}{30}(x-45) & (45\leqslant x<75)\\ 60-\frac{10}{15}(x-75) & (75\leqslant x<90)\\ 50+\frac{20}{15}(x-90) & (90\leqslant x<105)\end{cases}$$

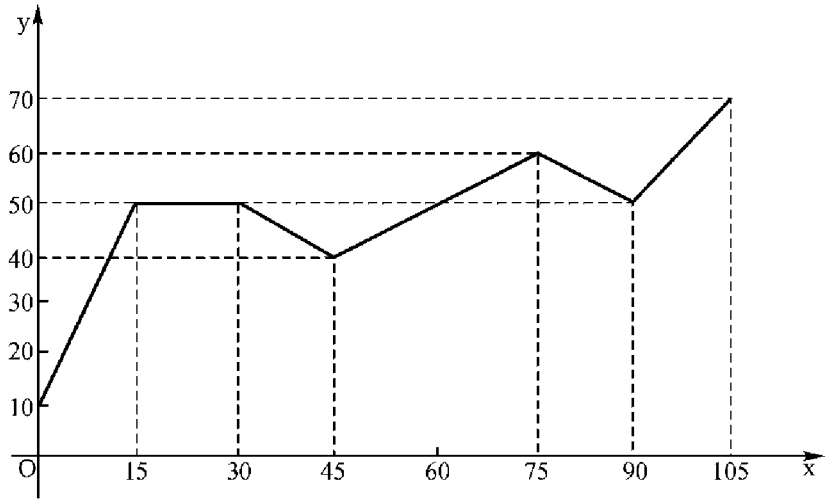


图 5.17

编写程序，要求在输入一个x值之后，打印出其相应的y值.

5. 为优待顾客，商店对购货额为1000元和1000元以上的，八折优待；500元以上（含500元下同）1000元以下的，九折优待；200元以上500元以下的，九五折优待；100元以上，200元以下的，九七折优待；100元以下，不优待. 请编出程序，在输入一个购货款额后，打印出该收的货款. 要求用ELSE IF语句.

6. 有一函数：

$$y=\begin{cases}\frac{y_2-y_1}{x_2-x_1}(x-x_1)+y_1 & (\text{当 } x_1\leqslant x<x_2)\\ \frac{y_3-y_2}{x_3-x_2}(x-x_2)+y_2 & (\text{当 } x_2\leqslant x\leqslant x_3)\end{cases}$$

输入x_1,x_2,x_3,y_1,y_2,y_3，再输入x，要求打印出相应的y值.

7. 有一方程 $Ax^2+Bx+C=0$，A，B，C 的值由键盘输入，请编写程序，打印出以下情况时方程的解.

（i）$A=0, B\neq 0$

（ii）$A=0, B=0, C=0$　　（x 为任何值）

（iii）$A=0, B=0$　$C\neq 0$　　（x 无解）

（iv）$A\neq 0, B^2-4AC>0$

（v）$A\neq 0, B^2-4AC=0$

（vi）$A\neq 0, B^2-4AC<0$

8. 以市中心为圆心，半径 20 公里以内（包括 20 公里），每亩地价 2 万元，20 公里以外的，每亩 1 万元. 请编程序，输入某一点的 x，y 坐标，求出该点的每亩地价（见图 5.18）.

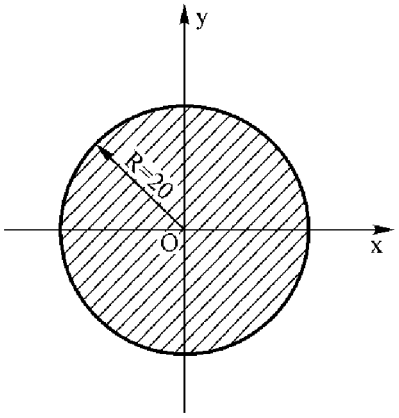


图 5.18

第七章　数　　组

迄今为止，我们已经知道在程序中一个变量代表了计算机内的一个存储单元；每个存储单元内，每次只能存放一个数据（整数、实数、逻辑值、字符串等）. 这种变量称为简单变量，每个简单变量都是独立的存储单元，彼此没有联系.

但是，在进行程序设计时，很多情况下只用简单变量是不够的. 例如，已知全区人口六万，假定居民中最高年龄为 120 岁，现在要求你编写一个程序来统计年龄的分布情况. 如果用简单变量 A1，A2，A3，…，A120 来分别统计各个年龄的总数，则需要写出如下的语句：

```
      READ( * , * ) AGE
10    IF( AGE. GE. 0) THEN
        IF( AGE. EQ. 1) A1 = A1 + 1
        IF( AGE. EQ. 2) A2 = A2 + 1
        IF( AGE. EQ. 3) A3 = A3 + 1

        IF( AGE. EQ. 120) A120 = A120 + 1
        READ( * , * ) AGE
        GOTO 10
      END IF

      WRITE( * , * ) A1, A2, A3, …, A120
```

从上面的举例可以看到，为了统计各种年龄的人数就需要 120 条逻辑 IF 语句，编写出来的程序将会十分繁琐冗长，显然，这是很不方便的.

为了解决这一类问题，许多高级语言都提供一种十分有用的数据结构——数组.

数组是由一系列数组元素组成. 一个数组中，所有的数组元素都用数组名作为名字，只是具有不同的下标，所以，通常也把数组元素称为下标变量. 例如，假定一个名字为 A 的数组由 120 个元素组成，则可以用 A(1)来表示 A 数组中的第一个元素，A(2)来表示 A 数组中的第二个元素，……，A(120)来表示 A 数组中第 120 个元素. 数组名后的一对括号中放的是下标.

在计算机内存中，每个数组占有一串连续的存储单元. 图 7.1 中示意了上述 A 数组各元素在内存中的分布情况. 每个存储单元都可以通过引用数组元素来直接进行存取.

现在，我们可以利用数组来改写以上程序段. 把 1 岁的人数放在 A(1)中，把 2 岁的人数放在 A(2)中，…，把 120 岁的人数放在 A(120)中，改写后的程序段表示如下：

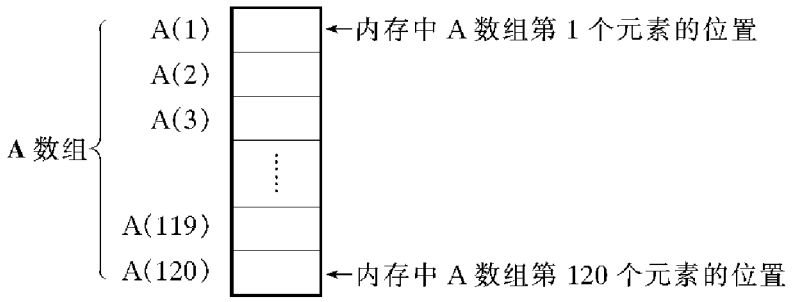


图 7.1

```
      READ( * , * ) AGE
10    IF( AGE. GT. 0) THEN
        DO 20 I = 1,120
          IF( AGE. EQ. I) A(I) = A(I) + 1
20      CONTINUE
        READ( * , * ) AGE
        GOTO 10
      END IF
      DO 30 AGE = 1,120
        WRITE( * , * ) AGE,' = ',A( AGE)
30    CONTINUE
      END
```

程序的输出如下所示：

```
 1 =   971   <-- AGE = 1 --   A(1)     971   1 岁的人数
 2 =   892   <-- AGE = 2 --   A(2)     892   2 岁的人数
 3 =   902   <-- AGE = 3 --   A(3)     902   3 岁的人数
 4 =   651   <-- AGE = 4 --   A(4)     651   4 岁的人数
 ⋮     ⋮                      ⋮       ⋮     ⋮
118 =    1   <-- AGE = 118 --  A(118)    1   118 岁的人数
119 =    0   <-- AGE = 119 --  A(119)    0   119 岁的人数
120 =    1   <-- AGE = 120 --  A(120)    1   120 岁的人数
```

每当输入一个年龄给 AGE 后，在 DO 循环中使 AGE 逐一和 1 到 120 进行比较. 如果输入的年龄为 5 岁，则当 I = 5 时，IF 语句中的条件满足，这时 A(I) = A(I) + 1 就相当于 A(5) = A(5) + 1，在 A 数组的第五个元素中，人数加 1. 这样，每读入一个年龄，就进行一次循环，使相应的 A 数组元素中的值加 1，从而达到了用 A 数组各元素来统计各种年龄人数的目的. 在标号为 30 的循环内是一条打印语句；循环变量 AGE 从 1 变化到 120，从而把 A 数组中第 1 个元素到第 120 个元素

中的值(即各种年龄的人数)顺序打印出来.

从这个例子可以看到,在程序设计中使用数组可大大提高程序的功能和灵活性.

§7.1 一维数组

用类型说明语句定义数组. 在 FORTRAN 程序中使用数组时,必须首先对数组进行定义. 通常可用类型说明语句来定义数组. 例如:

```
INTEGER X(0: 10),SCORE(1: 30)
REAL    TAB(-2: 2),Y(1832: 1840)
```

在前面统计各年龄人数的程序中,所用到的 A 数组也应该用以下语句先进行定义:

```
INTEGER A(1: 120)
```

和所有类型说明语句一样,定义数组的语句也都必须放在所有可执行语句和 DATA 语句(参看本章 §7.3)之前,PROGRAM 和 IMPLICIT 语句之后.

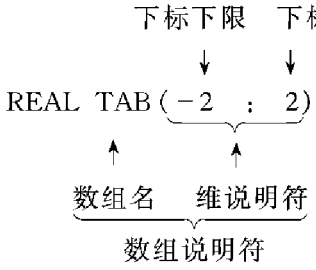
现在我们可以把统计各年龄人数的程序完善地写出如下:

```
      PROGRAM MAIN
      INTEGER A(1: 120),AGE
      DO 5 I=1,120                    }给 A 数组中每个元素赋初值零
        A(I)=0
5     CONTINUE
                                      }与前相同
      END
```

数组说明符. 在以上说明语句中,X(0: 10),SCORE(1: 30),A(1: 120),TAB(-2: 2),Y(1832: 1840)都称为数组说明符. 它们分别定义了在本程序单位中哪些名字是数组名,每个数组中各元素的下标个数和可用的下标范围. 在一条说明语句中可以有多个数组说明符,它们之间用逗号隔开. 在同一个程序单位中,一个数组说明符只能出现一次.

数组名的命名方法和简单变量名相同. 以上语句规定了 X,SCORE,A 都是整型数组名,TAB,Y 是实型数组名. 在同一个程序单位中,不允许数组名和简单变量名相同,也不允许一个数组名和另一个数组名相重.

维说明符. 在数组说明符中,放在数组名后一对括号内的是维说明符. 如:



一个维说明符由下标下限和下标上限组成,它们之间用冒号隔开. 以上的说明语句通知 FORTRAN 编译程序,在编译时为 TAB 数组保留如图 7.2 所示的五个连续的存放实型数的存储单元;此说明语句所在的程序单位中,TAB 名代表一个数组;因为只有一个维说明符,因此 TAB 的数组

图 7.2

元素只有一个下标，下标的范围为 -2 到 2.

用一个维说明符定义的数组称一维数组. 它的数组元素只能有一个下标. FORTRAN 77 规定，在数组说明符中，维说明符最多可达七个. 当有多个维说明符时，它们之间用逗号隔开. 例如：

```
REAL  S(0: 2,10: 20)
```

有两个维说明符，因此 S 是一个二维数组. 这将在本章第 4 节中介绍.

下标的下限和上限. 在主程序中，维说明符中的下标下限和上限只能用整常数或者整型符号常数所构成的表达式来表示（整型符号常数在本章 §7.3 中介绍），而不能用变量或算术表达式来表示. 例如：

```
REAL TAB( -2: 1 +1)
```

是正确的. 而

```
READ( *, * ) N,M
REAL TAB( N: M)
```

则是错误的. 因此在 FORTRAN 程序的主程序中，数组的大小（即数组元素的个数）是固定的，数组元素的下标下限和上限也是固定的，不可能在执行程序的过程中进行调节，要改变数组下标的下限和上限，必须修改程序.

当维说明符中的下标下限为 1 时，则可以简化维说明符. 例如可以把

```
INTEGER A(1: 120)
```

写成：

```
INTEGER A(120)
```

（请读者注意：FORTRAN 77 的某些子集只允许下标的下限为 1）.

DIMENSION 语句. 还可以用 DIMENSION 语句对数组进行定义. 例如：

```
INTEGER A(120)
```

可以用下面两条语句来代替：

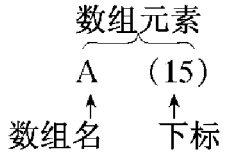
```
DIMENSION A(120)
INTEGER A
```

DIMENSION 只能定义数组不能说明数组类型. 当需要规定 A 数组是整型数组时，必须用 INTEGER 语句对数组名 A 进行说明，但不允许写成：

```
DIMENSION A(120)
INTEGER   A(120)
```

如果只用 DIMENSION 语句来说明数组，则由 FORTRAN 类型隐含规则来确定数组名是实型或是整型.

数组元素. 每个数组元素占一个存储单元，它是一个可以单独引用的变量. 通常，它可出现在任何允许简单变量出现的地方. 它的表示形式为：



在 FORTRAN 77 中，下标允许用任意合法的算术表达式. 当表达式的值为实数时，自动截取整数部分. 但下标的值必须落在数组说明中所规定的范围内，否则会导致出错，而这种错误往往是在程序执行的过程中出现的，因此很难查找.

同一数组中的所有数组元素类型相同.

例 **1**　输入 20 个整数，并按输入时的逆序打印出来，每行打印 5 个数.

程序如下所示.

```
      INTEGER A(20)
      DO 1 I=1,20
         READ(*,*) A(I)
1     CONTINUE
      DO 2 K=20,1,-5
         WRITE(*,*) A(K),A(K-1),A(K-2),A(K-3),A(K-4)
2     CONTINUE
      END
```

第一个 DO 循环中的 READ 语句一次读入一个数，因此需要 20 个输入行，每行一个数. 输入的数按输入的次序分别存放在 A(1)，A(2)，…，A(20)中. 第二个 DO 循环中是一条 WRITE 语句，在此语句的输出表中有 5 个数组元素，因此一行输出 5 个数. 循环控制变量从 20 变化到 1，步长为 -5. 第一次循环时，K = 20，循环中的 WRITE 语句相当于：

```
WRITE(*,*) A(20), A(19), A(18), A(17), A(16)
```

第二次循环时，K = 15，WRITE 语句就相当于：

```
WRITE(*,*)A(15),A(14),A(13),A(12),A(11)
```

以此类推，最后按输入时的逆序共分四行输出数据.

例 **2**　从 10 个整数中把最小的数找出来，并与第一个位置上的数对调，指出最小的数原来在数列中的位置.

为了说明简单起见，我们假定顺序读入 8，3，5，2，1，9 这六个数到数组中. 用变量 L 来记录数组中最小的那个数所在的位置(即下标)，在进行比较之前，先假设 L = 1. 以下说明中画圈者表示当前找到的最小数，带下划线者表示将要和最小数进行比较的数. 现将算法解释如下：

开始比较之前	L=1	⑧	3	5	2	1	9	把 A(L)与 A(2)比较
第一次比较之后	L=2	8	③	5	2	1	9	把 A(L)与 A(3)比较
第二次比较之后	L=2	8	③	5	2	1	9	把 A(L)与 A(4)比较
第三次比较之后	L=4	8	3	5	②	1	9	把 A(L)与 A(5)比较
第四次比较之后	L=5	8	3	5	2	①	9	把 A(L)与 A(6)比较
第五次比较之后	L=5	8	3	5	2	①	9	比较结果最小数在 A(L)之中.

框图如图 7.3 所示,程序表示如下:

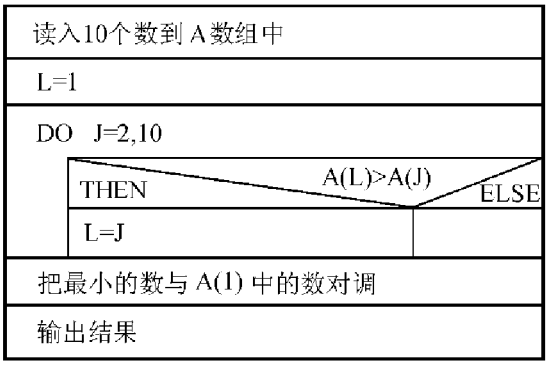


图 7.3

```
      PROGRAM T2
      INTEGER A(10)
      DO 1 I=1,10
         READ(*,*) A(I)
1     CONTINUE
      L=1
      DO 2 J=2,10
         IF(A(L).GT. A(J)) THEN
            L=J
         END IF
2     CONTINUE
      I=A(L)
      A(L)=A(1)
      A(1)=I
      WRITE(*,*) 'MIN NUMBER IS: ', A(1)
      WRITE(*,*) 'THE POSTION IS: ', L
      END
```

当输入 8,3,5,2,1,9,10,4,6,7 十个整数时,程序运行结果如下(分十行输入,每行一个数):

```
MIN MUMBER IS:    1
THE POSTION IS:   5
```

例 **3**　写一个程序,输入 10 个学生的成绩,把高于平均分的学生学号和成绩打印出来.

用 NUM 数组来存放学生学号,S 数组来存放对应的学生成绩. 变量 AVER 存放平均分. 算法和程序如下:

读入学号和学生成绩
求出平均成绩
把每个学生的成绩与平均分比较,当高于平均分时, 打印出学生学号和成绩

```
DIMENSION NUM(10), S(10)
SUM=0
```

```
      DO 20 I=1,10
         READ(*,*) NUM(I),S(I)
         SUM=SUM+S(I)
20    CONTINUE
      AVER=SUM/10.
      WRITE(*,200) AVER
200   FORMAT(1X,'A MEAN OF SCORES=',F6.2)
      WRITE(*,210)
210   FORMAT(//1X,'LIST OF SCORES GREATER THAN MEAN:'/)
      DO 30 I=1,10
         IF(S(I).GT.AVER) WRITE(*,*)NUM(I),S(I)
30    CONTINUE
      END
```

当输入以下数据时：

学　号	101	102	103	104	105	106	107	108	109	110
成　绩	86	93	77	62	81	90	75	69	87	94

程序运行的结果如下所示：

```
 A MEAN OF SCORES=81.40

 LIST OF SCORES GREATER THAN MEAN:

      101        86.0000000
      102        93.0000000
      106        90.0000000
      109        87.0000000
      110        94.0000000
```

§7.2　一维数组的输入和输出

数组的输入和输出可采用三种方式：

1. 把输入或输出语句放在 DO 循环中，利用 DO 循环输入或输出数组元素.

2. 在输入或输出语句中用数组名来输入或输出整个数组中的元素.

3. 在输入或输出语句中利用隐含 DO 循环来输入或输出数组中的元素.

本节将对这三种方式分别进行叙述.

利用 **DO** 循环进行输入或输出. 在此之前我们所举的例子中全都是把输入或输出语句放在 DO 循环中，利用循环控制变量作为数组元素的下标来控制数组元素的输入或输出. 例如：

```
      INTEGER A(50)
```

```
      DO 10 I=1,50
        READ(*,100) A(I)
10    CONTINUE
100   FORMAT(5I7)
```

以上语句允许把50个数按输入的顺序逐个放入A(1),A(2),…,A(50)中.但是应当注意,循环中的READ语句执行了50次,每次只能给A数组读入一个数,因为按规定,每执行一次READ语句就要求一个输入行,因此以上语句执行50次总共需要输入50行,每行一个数.同样,以下DO循环执行的结果是输出50行,每行输出一个数.

```
      DO 20 I=1,50
        WRITE(*,200) A(I)
20    CONTINUE
200   FORMAT(1X,5I7)
```

若要一行输出五个数,则必须在输出语句中写出五个数组元素.例如:

```
      DO 20 I=1,50,5
        WRITE(*,200) A(I),A(I+1),A(I+2),A(I+3),A(I+4)
20    CONTINUE
```

如果说明了一个大数组,而实际只需使用其中的一部分元素时,可以用以下语句:

```
      INTEGER A(50)
      READ(*,*) N
      DO 10 I=1,N
        READ(*,*) A(I)
10    CONTINUE
```

当N的值为15时,则READ语句执行15次,给A(1)到A(15)的元素输入值.数组中的其余元素(A(16)到A(50))无定义.

利用数组名进行输入输出.数组名(注意:不带下标)可以出现在输入输出语句中.例如:

```
      INTEGER KEEP(20)
      READ(*,*) KEEP
```

以上READ语句相当于:

```
      READ(*,*) KEEP(1),KEEP(2),KEEP(3),…,KEEP(20)
```

此语句执行一次,要求按表控格式输入20个数,计算机将按输入的顺序依次把数放在KEEP(1),KEEP(2),…,KEEP(20)中.

也可以用格式输入来控制每行输入数值的个数.例如:

```
      READ(*,100) KEEP
100   FORMAT(5I4)
```

此READ语句执行一次,要求输入20个数,但是按格式规定每行只能包含5个数.第一行的数依次赋给KEEP(1),…,KEEP(5);第二行的数依次赋给KEEP(6),…,KEEP(10);其余依次类推.注意,READ语句只执行了一次!

也可以用相似的方式输出数组中各元素的值. 例如：

```
WRITE( *,'(1X,5I5)') KEEP
```

以上 WRITE 语句执行一次，输出 20 个数，分四行输出，每行 5 个数.

用这种方式输入输出数组各元素的值显然比在 DO 循环中一次又一次地重复执行输入或输出语句效率要高得多，并且可以利用 FORMAT 语句来控制每行输入或输出数值的个数. 但缺点是，不管是否需要，在输入时必须给数组的全部元素输入值；在输出时，也必定会输出数组全部元素的值. 例如，如果给 X 数组定义了 100 个元素，如果我们写成：

```
WRITE( *,200) X
```

必定会输出 X 数组中 100 个元素的值，而不可能只输出 X(1)到 X(10)的值，也不可能只输出 X(2),X(4),X(6),…,X(100).

利用隐含 **DO** 循环进行输入和输出. 在输入输出语句中使用隐含 DO 循环将给输入和输出数组中的元素提供一种非常灵活的手段. 它可以按要求来指定数组中需要输入或输出的部分，并且可以人为地规定输入或输出的格式.

隐含 DO 循环的形式如下：

$$(\mathbf{list}, \mathbf{v}=\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3) \quad 或 \quad (\mathbf{list}, \mathbf{v}=\mathbf{e}_1, \mathbf{e}_2)$$

左括号相当于 DO，右括号相当于与之对应的 CONTINUE. 右括号前的 v, e_1, e_2, e_3 相当于 DO 循环中的循环控制变量、循环初值、循环终值和循环步长；当步长为 1 时 e_3 和它之前的逗号可省略不写. list 代表输入或输出表项，凡是可在输入输出语句中出现的合法的输入输出项都可出现在 list 中，因此 list 中也可以包括隐含 DO 循环. 以下是包含隐含 DO 循环的输出语句：

```
      INTEGER X(20)

      WRITE( *,100) (X(I),I=1,20)
100   FORMAT (1X,5I5)
```

以上 WRITE 语句相当于：

```
WRITE( *,100) X(1),X(2),…,X(20)
```

但是，利用隐含 DO 循环可以只输出 X 数组中的部分元素的值，例如：

```
WRITE( *,100) (X(I),I=1,10)
```

它相当于：

```
WRITE( *,100) X(1),X(2),…,X(10)
```

又如：

```
WRITE( *,100) (X(I),I=1,20,2)
```

相当于：

```
WRITE( *,100) X(1),X(3),X(5),…,X(19)
```

当只需要给 X 数组中的前 N 个元素输入值时，可以写成：

```
READ( *, *) N,(X(I),I=1,N)
```

但必须注意，N 不能放在隐含 DO 循环内. 以下语句是错误的：

```
READ( *, * ) (N,X(I),I=1,N)
```

因为在进入循环之前，循环终值 N 必须有确定的值，而以上语句却规定了 N 要在进入循环之后方能由输入的值来确定，因此是错误的.

以下简单列举了在输入输出语句中使用隐含 DO 循环的情况.

```
      INTEGER S(100)
      READ( *, * ) N, (S(I),I=1,N)
      WRITE( *,100) N, (I,S(I),I=1,N)
100   FORMAT(5X,'LIST OF',I4,'SCORES:'//(7X,I4,4X,I6))
      END
```

输入数据的形式：

```
6,85,90,92,63,77,100
```

输出结果：

```
LIST OF 6 SCORES:

     1          85
     2          90
     3          92
     4          63
     5          77
     6         100
```

§7.3 PARAMETER 语句和 DATA 语句

7.3.1 PARAMETER 语句

PARAMETER 语句是一种说明语句，用来指定本程序单位中的某些名字为符号常数. 符号常数的作用和普通常数的作用相同. 语句形式如下：

PARAMETER $(p_1 = c_1, p_2 = c_2, \cdots, p_n = c_n)$

这里，$p_1, p_2, \cdots, p_n$ 是符号常数名，它的命名方式和简单变量完全相同，类型说明也相同. $c_1, c_2, \cdots, c_n$ 是常数和符号常数（已定义过的）所组成的常数表达式. 例如：

```
      REAL      FACTOR,SEC
      PARAMETER (FACTOR=0.05)
      PARAMETER (SEC=FACTOR*5.29876,PI=3.141592)
```

PARAMETER 语句是非执行语句，它必须放在与它有关的类型说明语句之后，所有可执行语句之前. 在程序单位中，一个名字一旦用 PARAMETER 语句定义成符号常数，就不能再接受赋值，如以下语句是错误的：

```
      PARAMETER (N=10)
      READ( *, * ) N
```

符号常数不能在 FORMAT 语句中使用，不能用作语句标号，也不能出现在复型常数中（见§11.1）. 除此之外，可以在任何使用普通常数的地方使用它. 因此，在数组说明语句中的维说明符中，可以用符号常数来表示下标的下界和上界. 例如第 141 页例 3 中的程序可修改成：

```
      PARAMETER (N=10)
      DIMENSION NUM(N),S(N)
      SUM=0.0
      DO 20 I=1,N

20    CONTINUE
      AVER=SUM/N
      WRITE(*,200)N,AVER

      DO 30 I=1,N

30    CONTINUE
      END
```

这时，程序如果改成读入 30 个学生的成绩时，就只要简单地修改一条 PARAMETER 语句，使 N = 30，其他语句都不必改动，因而避免了由于修改中的疏忽而导致出错. 这是 PARAMETER 语句的主要用途之一.

PARAMETER 语句中所定义的符号常数，可以是整型、实型、双精度型、复型、逻辑型和字符型.

7.3.2 DATA 语句

DATA 语句用来在编译期间给变量置初值. 因此，它是非执行语句. DATA 语句的形式如下：

DATA $\mathbf{list_1}$,/$\mathbf{data_1}$/,$\mathbf{list_2}$/$\mathbf{data_2}$/,…,$\mathbf{list_n}$/$\mathbf{data_n}$/

在这里，$list_i$ 可包含变量名、数组名、数组元素和隐含 DO 循环，它们之间用逗号隔开；$data_i$ 代表一个常数表，这些常数作为初值分配给 $list_i$ 中的各个变量. 例如：

```
      DATA A,B,C,D,E/0.0,0.0,0.0,3.1415,1.0/
```

这条语句也可以写成：

```
      DATA A/0.0/,B/0.0/,C/0.0/,D/3.1415/,E/1.0/
```

以上 DATA 语句给 A，B，C 都赋初值 0.0，因此可以写成：

```
      DATA A,B,C,D,E/3*0.0,3.1415,1.0/
```

注意：此语句的 3*0.0 中，“*”号不是代表乘号，而是意味着“三个 0.0”.

DATA 语句可放在有关的类型说明语句、DIMENSION 和 PARAMETER 语句之后，程序中的任何位置上. 但最好养成放在所有可执行语句之前的习惯.

注意：在同一程序单位中，一个变量不可能赋两次初值. 也就是说，一个变量只可能有一个初值. 如：

```
      DATA K/0/
      K = K + 1
      DO 10 I = 1, N
         K = K + 1

10    CONTINUE
100   WRITE( * , * ) K
      DATA K/10/
```

很多初学者认为以上程序一开始执行时 K 的初值为 0，在执行到 100 语句之后，又重新给 K 赋以 10 作为初值. 这是由于把 DATA 语句看成了可执行语句而出现的错误概念. 实际上，DATA 语句是非执行语句，变量 K 的初值是在编译期间给定的，而且只可能有一个初值. 由于在两条语句中给 K 赋初值，FORTRAN 编译程序取后者作为 K 的初值. 因此在程序一开始运行时，K 的初值是 10 而不是 0. 由此例可以看到，由于错把 DATA 语句当成执行语句，将导致错误的计算结果，这种错误比较隐蔽，不易发现.

可以利用 DATA 语句给整个数组赋初值，例如：

```
DIMENSION NUM(100)
DATA SUM, NUM/0.0, 50 * 0, 50 * 10/
```

以上 DATA 语句给变量 SUM 赋初值 0.0；给 NUM 数组的前 50 个元素赋初值 0，给后 50 个元素赋初值 10.

也可以利用隐含 DO 循环给数组中的部分元素赋初值. 如：

```
PARAMETER ( N = 30, M = 2 * N, N2 = N/2 )
DIMENSION B( N ), A( M ), C( 3 )
CHARACTER C
LOGICAL YES
DATA ( B( I ), I = 1, N, 2 )/N2 * 0.0/, ( A( I ), I = 1, 10 )/10 * -1.0/
DATA C/'A', 'B', 'C'/, YES/.TRUE./
```

以上第一条 DATA 语句给 B(1)，B(3)，…，B(29)赋初值 0.0，给 A 数组的前 10 个元素赋初值 -1.0(注意负数不应该用括号括起来)；第二条 DATA 语句给 C(1)，C(2)，C(3)分别赋字符 A，B，C，给 YES 赋. TRUE. .

在 DATA 语句中，初值的个数与变量的个数，在数量上必须一一相同，在类型上必须一一对应.

§7.4 多维数组

具有一个下标的数组元素所组成的数组称为一维数组. 具有两个下标的数组元素所组成的数组称为二维数组，其他依此类推. FORTRAN 77 规定数组的维数最多可用到七维，因此数组元素的下标最多可有七个. 通常，我们把二维和更多维数的数组统称为多维数组. 本节将着重讨论

二维和三维数组.

二维和三维数组的定义.和定义一维数组一样,可以用类型说明语句或 DIMENSION 语句定义多维数组.例如:

```
DIMENSION MAX(1:2,1:5),R(1:2,1:4,1:3)
```

此语句定义了名字 MAX 和 R 是数组名.在本程序单位中,MAX 是一个整型二维数组,它的数组元素可以有两个下标,第一个下标的范围从 1 到 2,第二个下标的范围从 1 到 5;R 是一个实型三维数组,它的数组元素可以有三个下标,它们的下标范围分别为 1 到 2、1 到 4 和 1 到 3.以上 DIMENSION 语句中,各个维说明符中的下标下限都为 1,因此语句可写成:

```
DIMENSION MAX(2,5),R(2,4,3)
```

如果用类型说明语句来定义这两个数组,则可改写成:

```
INTEGER MAX(2,5)
REAL R(2,4,3)
```

二维和三维数组的数组元素.数组元素中的下标数与维数相同.也就是说,二维数组的元素有两个下标,三维数组的元素有三个下标.各下标之间用逗号隔开.如:MAX(1,2),MAX(2,5),R(2,4,1),R(2,1,2).

二维和三维数组的逻辑结构和存储结构.在逻辑结构上可以把一维数组看成是一个有序的表列,把二维数组看成是一张表格.以上定义的 MAX 数组,具有以下逻辑结构:

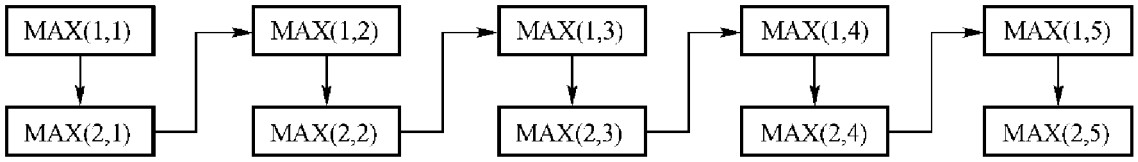
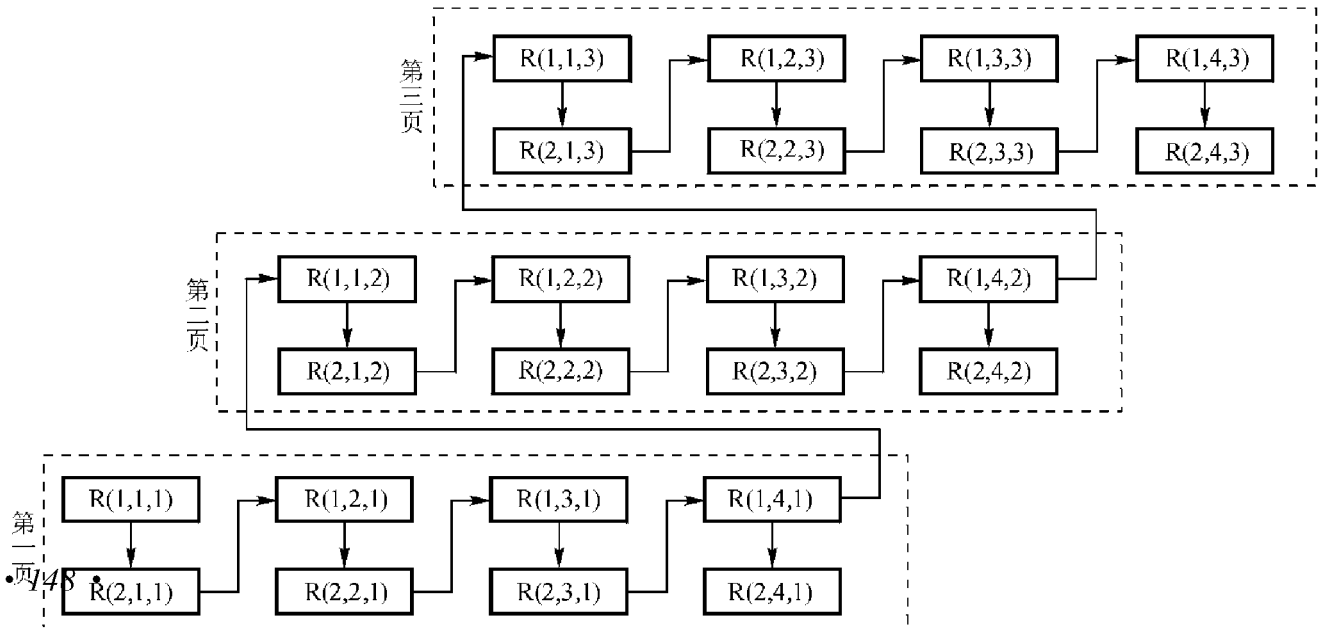


图 7.4

数组元素的第一个下标可代表此元素在 MAX 数组中的行号,而第二个下标则代表此元素所在的列号.例如数组元素 MAX(2,3)代表 MAX 数组中第二行、第三列上的元素.

三维数组可以看成是由若干页的表格所组成.以上定义的 R 数组,其结构可以形象地表示如下:



R 数组元素的第三个下标代表此元素所在的页号，而第一、第二个下标则分别代表所在页上的行号和列号. 如数组元素 R(2,1,3)代表在 R 数组第三页上，第二行、第一列上的元素.

在图 7.4 和 7.5 中，箭头表示了 MAX 和 R 数组中各元素在计算机内存中的存放次序. 前面曾经提到过，实际上，每个数组在内存中占用一串连续的存储单元，也就是说，每个数组的数组元素在内存中总是一个接着一个存放的. 如以上 MAX 数组一共需要 10 个连续的存储单元，R 数组需要 24 个连续的存储单元，它们在内存中的存放形式如图 7.6 所示. 对于二维数组来说，FORTRAN 规定先存放第一列上的元素，再存放第二列上的元素，…. 对于三维数组来说，先存放第一页中的元素，再存放第二页中的元素，逐页存放；每一页中也总是先放第一列元素，再放第二列元素，…. 通常把这种存放方式称为“按列存放”. 读者应该清楚地了解 FORTRAN 中数组在内存中存放的次序，这对正确设计程序以便得到预期的结果是非常重要的.

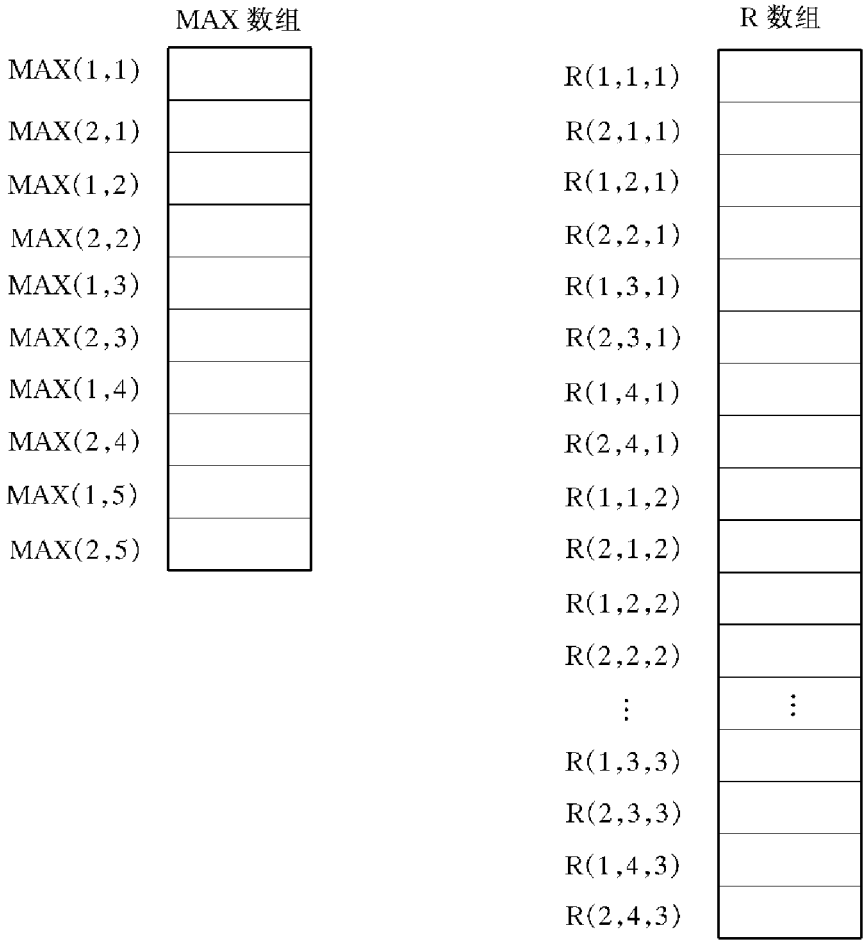


图 7.6

二维数组的输入输出. 同样，也可以利用三种方式输入输出多维数组元素的值. 在此，只以二维数组为例进行说明，掌握了这些基本内容，也就不难对多维数组设计输入和输出了.

假定已定义了 M 为一个二行三列的数组：

```
INTEGER M(2,3)
```

数组中的值如图 7.7 所示. 可以利用以下 DO 循环来输出 M 数组中的值.

```
      DO 30 I = 1,2
        DO 40 J = 1,3
          WRITE( * ,100) M(I,J)
40      CONTINUE
30    CONTINUE
100   FORMAT(1X,3I4)
```

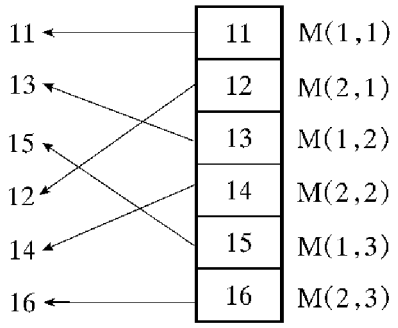


图 7.7

输出结果如图 7.7 中箭头所示.

在以上嵌套 DO 循环中的 WRITE 语句执行六次才能把整个数组中的元素输出完毕，每执行一次 WRITE 语句输出一行，每行一个数. 可以看到 WRITE 语句每次只引用了与其对应的 FORMAT 语句中的一个 I4 编辑描述符，多余的弃之不用.

如果改用以下 DO 循环语句来输出 M 数组元素的值：

```
      DO 30 I = 1,3
        DO 40 J = 1,2
          WRITE( * ,100) M(J,I)
40      CONTINUE
30    CONTINUE
```

产生以下的输出结果（如图 7.8 所示）：

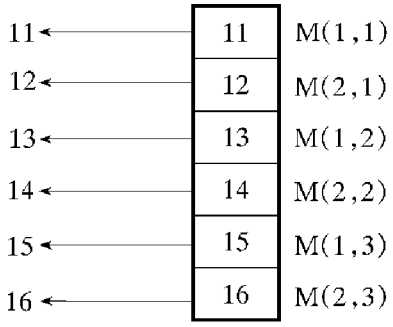


图 7.8

比较以上两次输出结果，可以看到，M 数组各元素输出的次序，完全由程序中所设计的 DO 循环来决定的.

当然，也可以通过在输入输出语句中写数组名的方式来输入或输出整个数组中的元素. 例如：

```
      INTEGER M(2,3)

      WRITE( * ,100) M
100   FORMAT(1X,3I4)
```

这时，不管 M 是几维数组，也不管 M 中包含有多少个元素，此 WRITE 语句执行一次将必定输出 M 数组中全部数组元素的值. 数组元素输出的次序与内存中该数组的元素排列次序相同. 至于具体的输出格式，可由所引用的 FORMAT 语句来规定. 根据以上定义，M 数组共有六个元素，WRITE(*,100)M 语句，相当于：

```
WRITE( *,100)M(1,1),M(2,1),M(1,2),M(2,2),M(1,3),M(2,3)
```

因此将按以上顺序输出六个元素的值，根据所引用的 FORMAT 语句的规定，它们将分两行输出，每行三个数，输出结果如图 7.9 箭头所示.

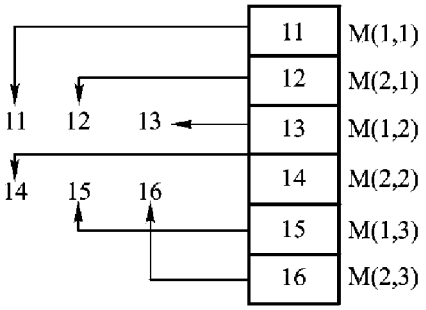


图 7.9

这个输出结果与数组的逻辑结构完全不同，这点务必请初学者注意.

如果把以上 FORMAT 语句改成：

```
      WRITE( *,100) M
100   FORMAT(1X,8I4)
```

则有以下输出形式（如图 7.10 所示）：

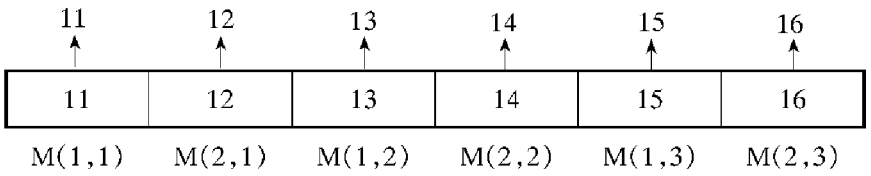


图 7.10

从以上例子可以看到，在 WRITE 语句中用数组名来输出整个数组时，总是按数组元素在计算机内存中排列的顺序来依次输出数组元素的值，而与数组的逻辑结构无关. 采用这种方式输出多维数组比起利用 DO 循环来输出数组元素，虽然能提高程序执行的效率，但却无法人为地控制数组元素输出的顺序.

同样，在 READ 语句中出现数组名时，也必须严格按照数组元素在计算机内存中排列的顺序输入数据，才能准确地给数组中各元素赋以正确的值. 例如，如果我们希望数组为以下矩阵时，则这些数据在内存中的排列应该如图 7.11 所示.

```
1   2   3
4   5   6
```

1	M(1,1)
4	M(2,1)
2	M(1,2)
5	M(2,2)
3	M(1,3)
6	M(2,3)

图 7.11

如用以下语句输入时：

```
      READ( *,200) M
200   FORMAT(3 I4)
```

输入数据必须按格式排列如下：

```
  1  4  2

  5  3  6
```

在输入输出语句中，利用隐含 DO 循环来输入输出多维数组，不仅具有较高的输入输出效率，而且能人为地控制数组中各元素输入输出的顺序和个数，因此具有较高的灵活性. 例如：

```
      DIMENSION NUM(3,4)
      READ( *,100)((NUM(I,J),J=1,3),I=1,2)
100   FORMAT(3 I4)
```

当输入数据为：

```
  1  2  3

  4  5  6
```

NUM 矩阵的左上角被赋值（如图 7.12），而这些数据在内存中的存放形式如图 7.13 所示. 如果需要给整个数组赋值时，READ 语句可改写成：

```
READ( *,100)((NUM(I,J),J=1,4),I=1,3)
```

此语句相当于以下语句：

```
      READ( *,100)NUM(1,1),NUM(1,2),NUM(1,3),NUM(1,4),
     *            NUM(2,1),NUM(2,2),NUM(2,3),NUM(2,4),
     *            NUM(3,1),NUM(3,2),NUM(3,3),NUM(3,4)
```

NUM 矩阵

1	2	3	
4	5	6	

图 7.12

1	NUM(1,1)
4	NUM(2,1)
	NUM(3,1)
2	NUM(1,2)
5	NUM(2,2)
	NUM(3,2)
3	NUM(1,3)
6	NUM(2,3)
	NUM(3,3)
	NUM(1,4)
	NUM(2,4)
	NUM(3,4)

图 7.13

具体的输出格式由所引用的 FORMAT 语句来规定.

§7.5 程序举例

例 **1** 规定学生成绩在90分以上为1级,75到89分为2级,60到74分为3级,60分以下为4级. 输入学生的学号和成绩. 打印出学生的学号和成绩的级别,并统计各种等级的学生人数.

此题的算法由图7.14表示. 程序中把评分标准用 DATA 语句放在 GRAD 数组中,由 NUM 数组各元素统计1到4级的人数,当输入数据小于0时表示输入数据结束. 二维数组 T 的第一列放学号,第二列上放对应学生的级别. 程序如下:

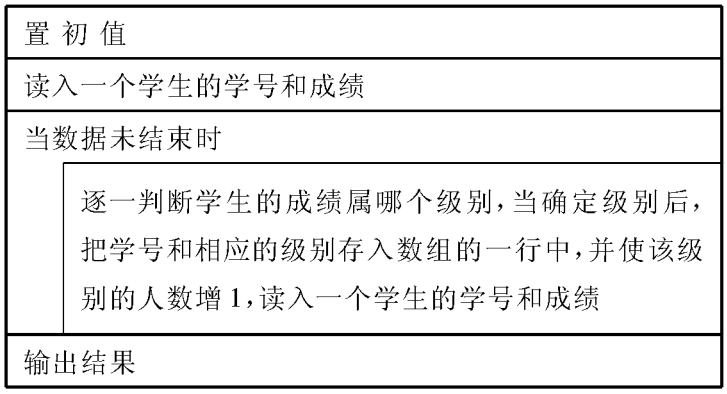


图 7.14

```
      INTEGER T(10,2),NUM(4)
      DIMENSION GRAD(5)
      DATA GRAD/101.0,90.0,75.0,60.0,0.0/,NUM/4*0/
      N=0
      READ(*,*) NS,S
2     IF(NS. GE. 0) THEN
          N=N+1
          DO 3 I=1,4
             IF(S. GE. GRAD(I+1). AND. S. LT. GRAD(I)) THEN
                 T(N,1)=NS
                 T(N,2)=I
                 NUM(I)=NUM(I)+1
             END IF
3         CONTINUE
          READ(*,*) NS,S
          GOTO 2
      END IF
      WRITE(*,100)
      WRITE(*,110)((T(I,J),J=1,2),I=1,N)
      WRITE(*,200)
```

```
      DO 4 I=1,4
         WRITE(*,210) I,NUM(I)
4     CONTINUE
100   FORMAT(5X,'STUDENT NUM',4X,'LEVEL')
110   FORMAT(8X,I6,8X,I2,'L')
200   FORMAT(5X,'LEVEL',6X,'NO. STUD')
210   FORMAT(7X,I2,'L',8X,I6)
      END
```

当学号 1 到 10 的 10 个学生成绩分别为 67,78,96,68,88,84,92,69,72,77 时,输出结果如下:

STUDENT NUM	LEVEL
1	3 L
2	2 L
3	1 L
4	3 L
5	2 L
6	2 L
7	1 L
8	3 L
9	3 L
10	2 L

LEVEL	NO. STUD
1 L	2
2 L	4
3 L	4
4 L	0

例 2 某工厂生产了八种产品,由五位推销员负责推销,右边矩阵是某月份各推销员推销产品的数量图. 第一行表示第一种产品销售的数量,第二行表示第二种产品销售数量,…. 第一列代表第一位销售员推销的各种产品的数量,第二列代表第二位销售员推销的各种产品的数量,…,第五列代表第五位销售员推销的各种产品的数量. 要求根据输入数据打印出以下表格. 表格中最后一列表示各推销员推销的产品总数,表格中的最后一行表示各种

$$\begin{bmatrix} 3 & 3 & 0 & 2 & 5 \\ 0 & 2 & 0 & 1 & 5 \\ 10 & 1 & 0 & 1 & 7 \\ 0 & 1 & 0 & 0 & 6 \\ 3 & 3 & 2 & 2 & 0 \\ 0 & 4 & 6 & 8 & 0 \\ 0 & 4 & 8 & 0 & 0 \\ 2 & 1 & 4 & 3 & 5 \end{bmatrix}$$

NUM OF PRODUCT

SALESMAN	:	1	2	3	4	5	6	7	8	:	TOTAL
1	:	3	0	10	0	3	0	0	2	:	18
2	:	3	2	1	1	3	4	4	1	:	19
3	:	0	0	0	0	2	6	8	4	:	20
4	:	2	1	1	0	2	8	0	3	:	17
5	:	5	5	7	6	0	0	0	5	:	28
TOTAL	:	13	8	19	7	10	18	12	15		

产品销售总数.

此题的算法如下所示：

程序如下，在程序中定义了8行5列的SALE数组来存放以上数据. NUM数组存放每种产品的推销总数，EVERY数组存放每位推销员推销的产品总数.

```
      INTEGER SALE(8,5),NUM(8),EVERY(5)
      READ(*,10)((SALE(J,I),I=1,5),J=1,8)
10    FORMAT(5I4)
      DO 1 I=1,5
         EVERY(I)=0
         DO 2 J=1,8
            EVERY(I)=EVERY(I)+SALE(J,I)
2        CONTINUE
1     CONTINUE
      DO 3 I=1,8
         NUM(I)=0
         DO 4 J=1,5
            NUM(I)=NUM(I)+SALE(I,J)
4        CONTINUE
3     CONTINUE
      WRITE(*,20)(I,I=1,8)
      WRITE(*,30)(J,(SALE(I,J),I=1,8),EVERY(J),J=1,5)
      WRITE(*,40)(NUM(I),I=1,8)
20    FORMAT(27X,'NUM OF PRODUCT'/
     1       8X,'SALESMAN:  ',8I4,'TOTAL:  '/
     2       6X,51('-'))
30    FORMAT(9X,I4,3X,':',8I4,':  ',I4)
40    FORMAT(6X,51('-')/8X,'TOTAL :  ',8I4)
      END
```

例**3**　假定以上例题中产品的价格如下表所示：

产 品 号	1	2	3	4	5	6	7	8
单价(元)	20	50	80	100	120	150	180	200

求每位销售员在该月中的总推销额.

在程序中定义一个5行8列的二维数组SALE，每行中存放每位推销员推销的各类产品数.在一维数组PRICE中依次放入每种产品的单价.TOTAL数组中存放计算所得的每位销售员推销的总金额，它们的计算公式如下所示：

$$TOTAL(1)=PRICE(1)*SALE(1,1)+PRICE(2)*SALE(1,2)+PRICE(3)*SALE(1,3)+\cdots+PRICE(8)*SALE(1,8)$$

$$TOTAL(2)=PRICE(1)*SALE(2,1)+PRICE(2)*SALE(2,2)+PRICE(3)*SALE(2,3)+\cdots+PRICE(8)*SALE(2,8)$$

$$TOTAL(5)=PRICE(1)*SALE(5,1)+PRICE(2)*SALE(5,2)+PRICE(3)*SALE(5,3)+\cdots+PRICE(8)*SALE(5,8)$$

程序如下：

```
      INTEGER SALE(5,8)
      REAL    PRICE(8),TOTAL(5)
      READ(*,*) SALE,PRICE
      DO 2 I=1,5
         TOTAL(I)=0
         DO 1 J=1,8
            TOTAL(I)=TOTAL(I)+PRICE(J)*SALE(I,J)
1        CONTINUE
2     CONTINUE
      WRITE(*,100)(I,I=1,5)
100   FORMAT(2X,'NO. SALESMAN :  ',5I8/1X,57('-'))
      WRITE(*,200) TOTAL
200   FORMAT(5X,'TOTAL SALES:',5F8.2/1X,57('-'))
      END
```

输出结果如下：

```
      NO. SALESMAN      :        1         2         3         4         5
-------------------------------------------------------------------------------
        TOTAL SALES     :  1830.00   1490.00   2400.00   2150.00   4180.00
-------------------------------------------------------------------------------
```

例**4**　全年级六个班学生进行数学考试.阅卷后各班的考卷混在一起，但考卷上写明了班号和成绩.现在来编写一个程序，通过输入班号和成绩分别统计出各班的平均分.已知班号分别为8501～8506.

程序中用NUM数组来统计各班人数，ST数组存放各班总分，AV数组存放各班的平均分.程序如下：

```
      PROGRAM    T8
      DIMENSION AV(8501: 8506)
      DIMENSION NUM(8501: 8506),ST(8501: 8506)
      DO 1 I=8501,8506
```

```
          NUM(I)=0
          ST(I)=0
1         CONTINUE
          READ(*,*) N,SC
2         IF(N. GE. 0) THEN
              NUM(N)=NUM(N)+1
              ST(N)=ST(N)+SC
              READ(*,*) N,SC
              GOTO 2
          END IF
          DO 3 I=8501,8506
              AV(I)=ST(I)/NUM(I)
3         CONTINUE
          WRITE(*,100)
100       FORMAT(10X,'CLASS NUM',2X,'AVERAGE SCORE')
          WRITE(*,200)(I,AV(I),I=8501,8506)
200       FORMAT(13X,I4,8X,F8.4)
          END
```

程序在执行过程中，每次读入一个班号给 N，读入一个成绩给 SC，当输入班号为负数时表示输入的数据结束.

例 **5**　把任意个正整数插入到一个按升序排列的有序整数数列中. 一次插入一个，插入后的数列仍然有序.

程序中把有序的数列放在 A 数组中，要插入的整数放在变量 X 中，当 X 值得负时，表示插入结束. 变量 N 用来统计当前数列中数的个数. 算法如图 7.15 所示.

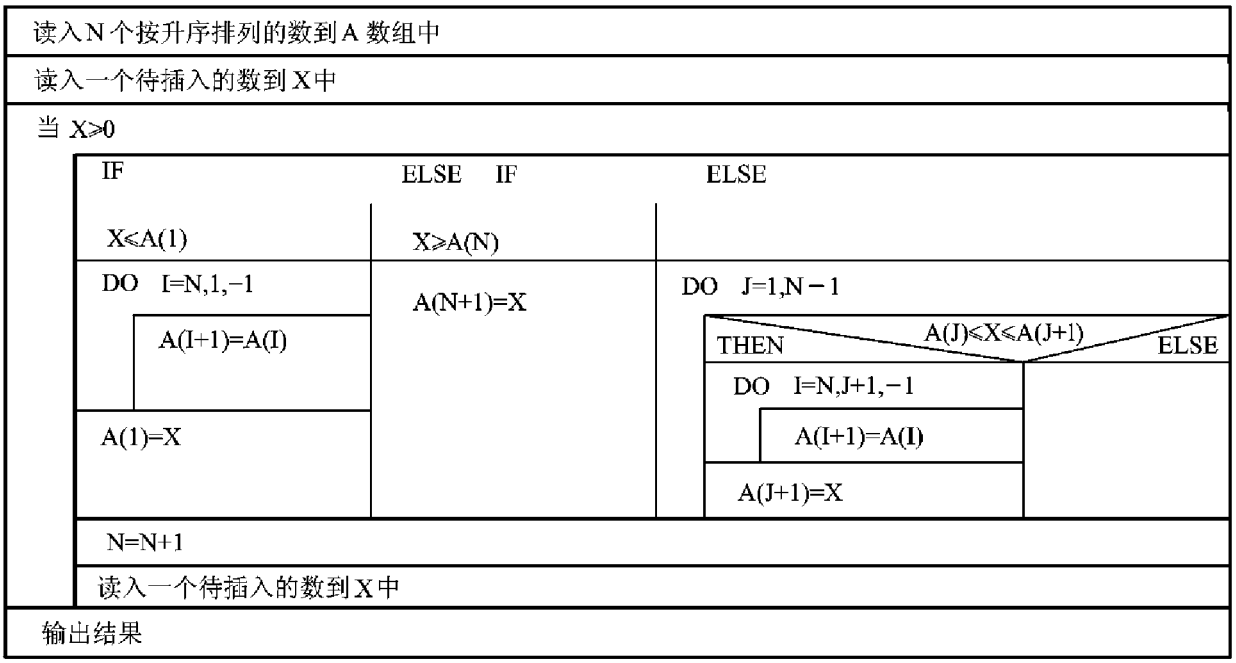


图 7.15

程序如下：

```
      PROGRAM    INSERT
      INTEGER A(100),X
      READ(*,*) N,(A(I),I=1,N)
      READ(*,*) X
10    IF(X. GE. 0) THEN
        IF(X. LE. A(1)) THEN
          DO 20 I=N,1,-1
            A(I+1)=A(I)
20        CONTINUE
          A(1)=X
        ELSE IF(X. GE. A(N)) THEN
            A(N+1)=X
          ELSE
            DO 40 J=1,N-1
              IF(X. GT. A(J). AND. X. LE. A(J+1)) THEN
                DO 30 I=N,J+1,-1
                  A(I+1)=A(I)
30              CONTINUE
                A(J+1)=X
              END IF
40            CONTINUE
          END IF
        END IF
        N=N+1
        READ(*,*) X
        GOTO 10
      END IF
      WRITE(*,100)(A(I),I=1,N)
100   FORMAT(1X,'THE NEW SEQUENCE:'/(1X,10I5))
      END
```

例 **6**　把两个按升序排列的数列合并成一个，合并后的数列仍按升序排列.

程序中用 A，B 数组存放待合并的数列. NA，NB 分别代表 A，B 数组中数的个数. IA，IB 分别代表当前待放入到 C 数组中去的 A，B 数组元素的下标. IC 指向 C 数组中当前存放数值的元素位置(下标). 算法框图如图 7.16 所示. 程序如下：

```
      INTEGER A(100),B(100),C(100)
      DATA IA,IB,IC/2*1,0/
      READ(*,*) NA,(A(I),I=1,NA)
      READ(*,*) NB,(B(I),I=1,NB)
10    IF(IA. LE. NA. AND. IB. LE. NB) THEN
```

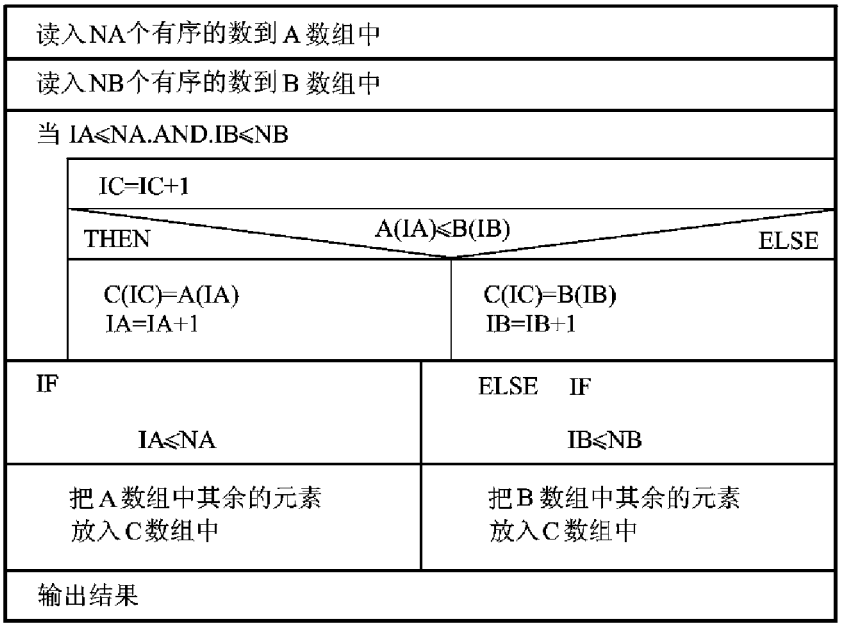


图 7.16

```
            IC = IC + 1
            IF( A( IA ). LE. B( IB ) ) THEN
                C( IC ) = A( IA )
                IA = IA + 1
            ELSE
                C( IC ) = B( IB )
                IB = IB + 1
            END IF
        GOTO 10
        END IF
        IF( IA. LE. NA ) THEN
            DO 20 J = IA, NA
                IC = IC + 1
                C( IC ) = A( J )
20          CONTINUE
        ELSE IF( IB. LE. NB ) THEN
            DO 30 J = IB, NB
                IC = IC + 1
                C( IC ) = B( J )
30          CONTINUE
        END IF
        WRITE( *, * )'A SEQ:', ( A( I ), I = 1, NA )
        WRITE( *, * )'B SEQ:', ( B( I ), I = 1, NB )
        WRITE( *, * )'C SEQ:', ( C( I ), I = 1, IC )
        END
```

当输入以下数据时：

4,1,3,5,7

5,0,2,4,7,8

得以下输出结果：

A SEQ:1 3 5 7

B SEQ:0 2 4 7 8

C SEQ:0 1 2 3 4 5 7 7 8

例 **7** 输入 N 个无序的数，编写程序，将此 N 个数按由小到大的顺序排好.

本章 §7.1 的例 2 中曾要求把数列中最小的数找出来，并与第一个位置上的数对调. 该程序已完成了 N 个数排序的第一步，以后只要再从第二到第 N 个元素之间去找最小的那个数并与第二个位置上的数对调，依此类推，直到从最后两个元素中找最小数，并把小的数放在前面，则整个数列的排序也就完成了. 因此排序的过程实际上就是重复这样的过程：从第 J 到第 N 个数中找出最小的那个数，并把此数与第 J 个位置上的数对调. 每重复一次，J 加 1，即范围向右推移一位，一直到 J = N - 1，即重复 N - 1 次为止. 程序框图见图 7.17，程序如下：

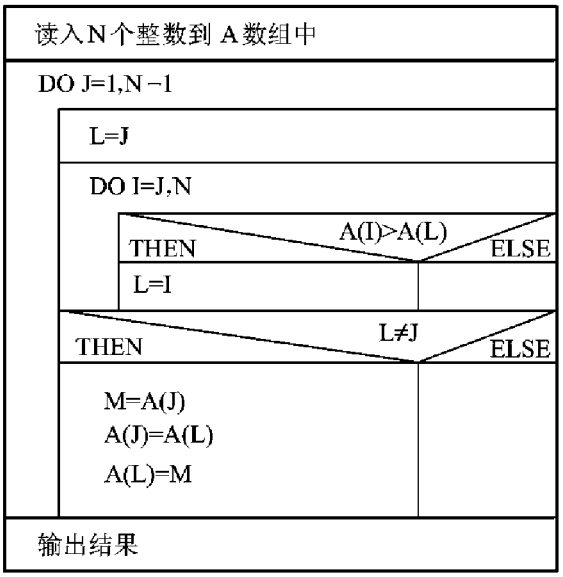


图 7.17

```
      INTEGER A(100)
      READ(*,*) N,(A(I),I=1,N)
      WRITE(*,*)'INPUT A SEC:'
      WRITE(*,100) (A(I),I=1,N)
      DO 20 J=1,N-1
         L=J
         DO 10 I=J,N
            IF(A(I).LT.A(L)) THEN
              L=I
            END IF
10       CONTINUE
         IF(L.NE.J) THEN
```

```
                M = A(J)
                A(J) = A(L)
                A(L) = M
            END IF
        WRITE( * ,100) (A(I),I = 1,N)
100     FORMAT(/1X,10I4)
20      CONTINUE
        WRITE( * , * ) 'A OUTPUT SEC:'
        WRITE( * ,100) (A(I),I = 1,N)
        END
```

当程序执行时输入：

```
6
8,3,5,2,1,9
```

得以下输出结果：

```
INPUT A SEC:
   8  3  5  2  1  9
   1  3  5  2  8  9
   1  2  5  3  8  9      这5行是在排序过程
   1  2  3  5  8  9      中输出的中间结果
   1  2  3  5  8  9
   1  2  3  5  8  9
A OUTPUT SEC:
   1  2  3  5  8  9
```

例 **8**　有六辆汽车同时到加油站加油. 每辆车加油所需时间分别为 $S_1, S_2, \cdots, S_6$. 问按什么顺序加油时，使所有汽车总共所花的等待时间最少.

如果先给加油时间为 S_1 的车加油，再给加油时间为 S_2 的车加油，依此类推，则六辆车总共所花的等待时间为：

$$W = 6 * S_1 + 5 * S_2 + 4 * S_3 + 3 * S_4 + 2 * S_5 + S_6$$

解决这个问题的最简单方法是穷举法，即去试验每一种可能的排队方案，比较哪种排队方案所花的排队时间最少.

现将算法解释如下：程序中用 S 数组分别存放六辆车的加油时间，用

$$W = 6 * S(K1) + 5 * S(K2) + 4 * S(K3) + 3 * S(K4) + 2 * S(K5) + S(K6)$$

来求得某种排队次序时总共所花的等待时间. K1, K2, …, K6 的值代表排在第 1 到第 6 的各辆车号，如当 K1 = 2, K2 = 4, K3 = 1, K4 = 5, K5 = 6, K6 = 21 - (K1 + K2 + K3 + K4 + K5) = 3，这时求得的 W 值即表示第 2 号车排在第 1，第 4 号车排在第 2，第 1 号车排在第 3，第 5 号车排在第 4，第 6 号车排在第 5，第 3 号车排在第 6 时总共花费的排队时间. 把 K1, K2, …, K6 作为各循环的控制变量，它们的值分别从 1 到 6 变化，而且每次互不相同，这样就可列举六辆车全部可能的排队次序，

从而求出不同的等待时间.用 W0 来存放当前所求得的最少等待时间,每当得到的一种等待时间小于 W0 时,就把此时间存入 W0 中,并把排在第 1 的车号存入 NUM(1)中,…,排在第 6 的车号存入 NUM(6)中.每得到一组 K1,K2,…,K6,就重复以上过程,最后在数组 NUM 中的是所花等待时间最少的车号次序.程序的框图如图 7.18 所示.程序如下:

```
      DIMENSION S(6),NUM(6)
      W0 = 0
      DO 10 I = 1,6
          READ( * , * ) S(I)
          W0 = W0 + I * S(I)
          NUM(I) = I
10    CONTINUE
      DO 60 K1 = 1,6
          DO 50 K2 = 1,6
          IF(K1. NE. K2) THEN
              DO 40 K3 = 1,6
              IF(K3. NE. K1. AND. K3. NE. K2) THEN
                  DO 30 K4 = 1,6
                  IF(K4. NE. K1. AND. K4. NE. K2. AND. K4. NE. K3) THEN
                      DO 20 K5 = 1,6
                      IF(K5. NE. K1. AND. K5. NE. K2. * AND. K5. NE. K3.
                      AND. K5. NE. K4) THEN
                          K6 = 21 - (K1 + K2 + K3 + K4 + K5)
                          W = 6 * S(K1) + 5 * S(K2) + 4 * S(K3) + 3 * S(K4) + 2 * S(K5) + S(K6)
                          IF(W. LT. W0) THEN
                              W0 = W
                              NUM(1) = K1
                              NUM(2) = K2
                              NUM(3) = K3
                              NUM(4) = K4
                              NUM(5) = K5
                              NUM(6) = K6
                           END IF
                       END IF
20                     CONTINUE
                   END IF
30                 CONTINUE
               END IF
40             CONTINUE
           END IF
50         CONTINUE
```

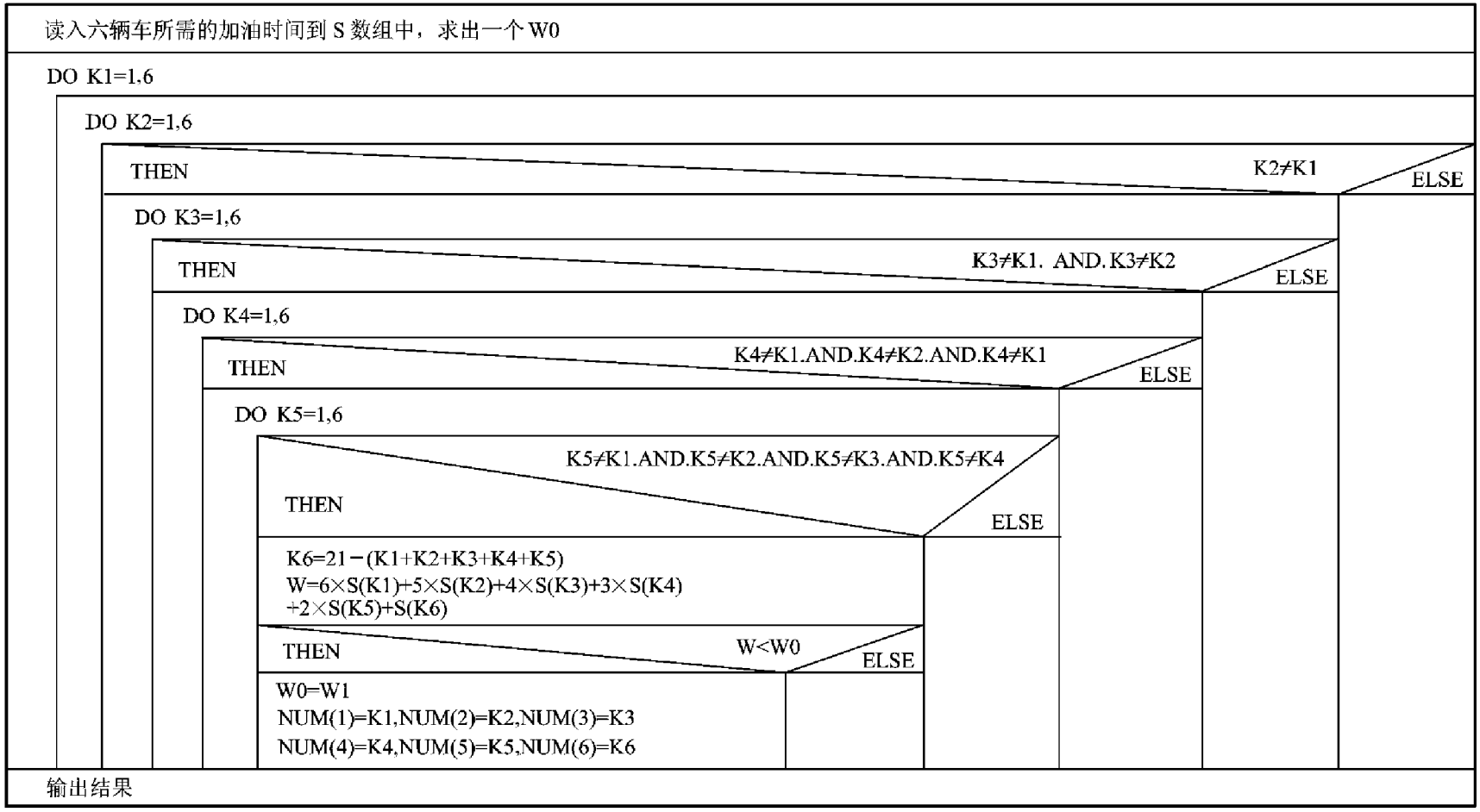

读入六辆车所需的加油时间到 S 数组中，求出一个 W0
DO K1=1,6
DO K2=1,6
THEN
K2≠K1
ELSE
DO K3=1,6
THEN
K3≠K1. AND. K3≠K2
ELSE
DO K4=1,6
THEN
K4≠K1.AND.K4≠K2.AND.K4≠K1
ELSE
DO K5=1,6
K5≠K1.AND.K5≠K2.AND.K5≠K3.AND.K5≠K4
THEN
ELSE
K6=21－(K1+K2+K3+K4+K5)
W=6×S(K1)+5×S(K2)+4×S(K3)+3×S(K4)
+2×S(K5)+S(K6)
THEN
W<W0
ELSE
W0=W1
NUM(1)=K1,NUM(2)=K2,NUM(3)=K3
NUM(4)=K4,NUM(5)=K5,NUM(6)=K6
输出结果

图 7.18

```
60        CONTINUE
          WRITE( *,100)(NUM(I),I=1,6)
          WRITE( *,200)(S(NUM(I)),I=1,6)
100       FORMAT(1X,'THE SEQUICE IS:',6I5)
200       FORMAT(1X,'THE TIME IS:',6F5.1,'MIMUTE')
          END
```

当第 1 到第 6 辆车加油的时间分别为:5 秒、4 秒、7 秒、12 秒、3 秒、8 秒时,输出结果如下所示:

```
THE SEQUICE IS:  5    2    1    3    6    4
THE   TIME   IS:3.0  4.0  5.0  7.0  8.0  12.0  MIMUTE
```

例 **9**　N 只猴子要选猴王,它们围成一圈,从 1 开始依次连续报数,凡所报的数能被 3 除尽时,报该数的猴子就从圈中退出,所剩最后一个为猴王.

此题的算法如图 7.19 所示.

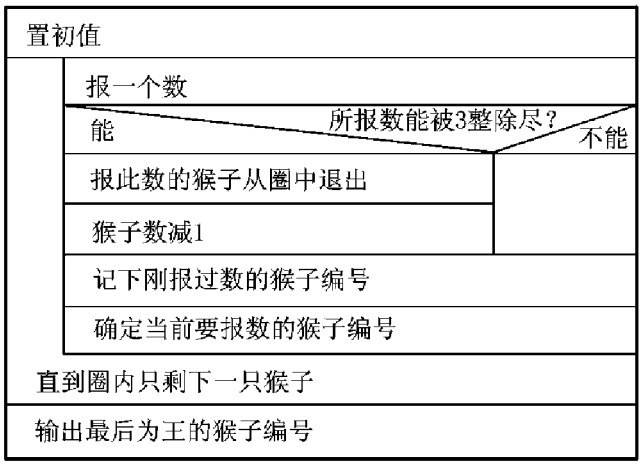


图 7.19

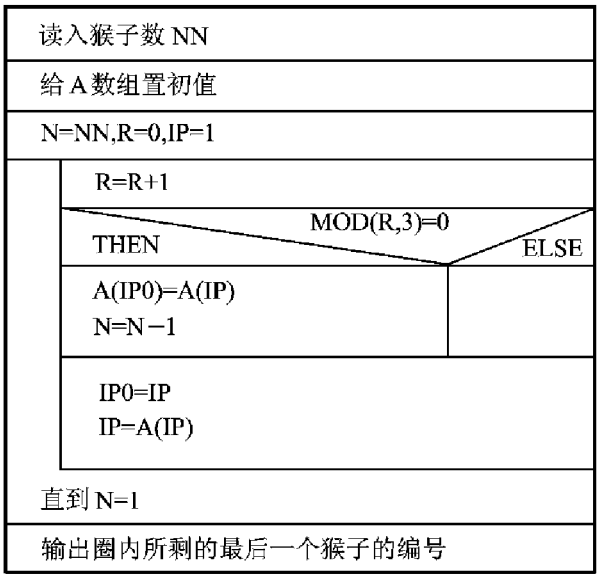


图 7.20

程序中,猴子报的数用 R 表示,变量 IP 指示当前报数的猴子号,IP0 指示刚报过数的猴子号. A 数组元素中存放下一个将报数的猴子编号,即,一开始,A(1) = 2,A(2) = 3,A(3) = 4,……,A(N) = 1. 变量 N 用来统计圈内尚剩的猴子数,把 3 存入变量 L.

当确定编号为 IP 的猴子从圈中退出时,使 A(IP0)中的编号改成 A(IP)中的号,这时 IP 当前所代表的编号就从 A 数组中消失了. 例如当从头开始报数时,当 R 等于 3 时,则使 A(2)⇐A(3),即把 A(3)中的 4 放入到 A(2)中,A(2)中原来的 3 消失;当 R 等于 6 时,使 A(5)⇐A(6),即把 7 放入 A(5)中,…. 这样下次轮到编号为 2 的猴子报完数后(这时 IP = 2),由于 A(2)中的值已改为 4,所以下一次报数的猴子号将为 4,而跳过了 3 号猴子. 程序的框图如图 7.20 所示. 按照框图写出的程序如下:

```
          INTEGER R,A(100)
          READ( *, * )NN,L
```

```
      N = NN
      DO 1 I = 1,N
         A(I) = I + 1
1     CONTINUE
      A(N) = 1
      R = 0
      IP = 1
2     R = R + 1
         IF(MOD(R,L). EQ. 0) THEN
            A(IP0) = A(IP)
            N = N - 1
         END IF
         IP0 = IP
         IP = A(IP)
      IF(N. EQ. 1) THEN
      ELSE
      GOTO 2
      END IF
      WRITE(*,*) 'THE KING IS:',IP
      WRITE(*,*) 'THE NUMBER OF MONKEY IS',NN
      WRITE(*,*) 'NUMBER L:',L
      END
```

当猴子数为13,报数号能被5整除的猴子从圈中退出时,输出结果如下:

```
THE KING IS:                    6          (第6号猴子为王)
THE NUMBER OF MONKEY IS                    13
NUMBER L:              5
```

当猴子数为20,报数号能被7整除的猴子从圈中退出时,输出结果如下:

```
THE KING IS:                    3
THE NUMBER OF MONKEY IS                    20
NUMBER L:              7
```

习　题

1. 在以下程序段中,A、B、C数组用几行输入?输出用几行?

```
      INTEGER A(50),B(50),C(50)
      DO 1 I = 1,50
         READ(*,100) A(I)
1     CONTINUE
      DO 2 I = 1,50
         WRITE(*,200) A(I)
2     CONTINUE
```

```
      READ( *,100) B
      READ( *,100) (C(I),I=1,30)
      WRITE( *,200) B
      WRITE( *,200) (C(I),I=1,20,2)
100   FORMAT(5I4)
200   FORMAT(1X,5I4)
      END
```

2. 有以下语句：

```
      DIMENSION X(3,4)
      INTEGER   X
      READ( *,'(4I3)')X
```

要求 X 数组中对应元素有以下的值：

77 56 32 25
99 10 100 46
48 89 77 33

根据以上 READ 语句写出输入数据的具体形式，再补充相应的一些语句，使程序输出以上形式的矩阵.

3. 输入 N 个数到数组中（N 最多为 100），选出所有大于 N 个数的平均值的那些数.

4. 输入 N 个数到数组中，选出其中最小的数和最大的数，并分别把它们放在第 1 和第 N 个位置上.

5. 分别输入 N 和 M 个数到 A 和 B 数组中，（A 和 B 数组的大小自定）. 把在 A 和 B 数组中都出现的那些数打印出来，并指出它们在 A 和 B 数组中的位置.

6. 分别输入 N 和 M 个数到 A 和 B 数组中. 把只在其中一个数组中出现的那些数打印出来，并指出它们在 A 和 B 数组中的位置.

7. 输入 N 个数到数组中，把出现次数最多的那个数找出来（可能有几个这样的数），并统计出现次数.

8. 假定有一叠卡片，卡片号为 1 到 150，并且所有卡片的正面都朝上. 从卡片号 2 开始，把凡是偶数的卡片都翻成正面朝下. 再从 3 号卡片开始，把凡是卡片号为 3 的倍数的卡片都翻一个面（即把正面朝上的翻成正面朝下，正面朝下的翻成正面朝上）. 下一步从 4 号卡片开始，把凡是卡片号为 4 的倍数的卡片都翻转一次，依次类推，写一个程序来测定全过程完成后，哪些卡片的面朝上，共有几张.

9. 用筛选法求 2 到 n 之间的素数. 方法如下：首先 2 是素数，而 2 的倍数都不是，于是把这些数从数表中筛去. 2 以后没被筛去的第一个数是 3，然后把 3 的倍数都从数表中筛去. 3 以后没被筛去的第一个数是 5，然后把 5 的倍数都从数表中筛去. 重复以上过程，直到遇到第一个未被筛去的数是$\sqrt{n}$ 为止. 这时保留下的未被筛去的数就都是素数.

10. 写一个程序，实现对任意长的两个大整数（例如 300 位）进行相加. 每个数可用以下形式存放，例如整数 179,534,679,198 可存放在数组中，其中 N(1) = 198，N(2) = 672，N(3) = 534，N(4) = 179. 把两个数组中的元素一一相加，并根据需要进行进位.

11. 写一个程序，使 6 × 6 的数组中，除对角线上的元素为 1 外，其余皆为 0.

12. 现有两个 m × n 的数组，写一个程序进行矩阵相加，（使两个数组中的元素一一对应相加）.

13. 写一个程序，找出矩阵中最大和最小的项和它们的位置.

14. 找出 m × n 数组中所有不相邻元素之和（以第一个元素为基准）.

15. 找出二维数组中，每一行中最大的那个元素.

16. 求出一个二维数组中每行元素的平均值.

17. 在一个二维数组中找出符合以下条件的元素，此元素在所在行为最大，在所在列为最小（也可能没有这样的元素，这时打印出相应的信息）.

18. 写一个程序，打印出以下形式的杨辉三角形.

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10 10 5  1
1  6  15 20 15 6  1
1  7  21 35 35 21 7  1
1  8  28 56 70 56 28 8  1
1  9  36 84 126 126 84 36 9  1
```

19. 写一个程序，打印出以下乘法九九表.

* * A MULTIPLICATION TABLE * *

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(1)	1	2	3	4	5	6	7	8	9
(2)	2	4	6	8	10	12	14	16	18
(3)	3	6	9	12	15	18	21	24	27
(4)	4	8	12	16	20	24	28	32	36
(5)	5	10	15	20	25	30	35	40	45
(6)	6	12	18	24	30	36	42	48	54
(7)	7	14	21	28	35	42	49	56	63
(8)	8	16	24	32	40	48	56	64	72
(9)	9	18	27	36	45	54	63	72	81

20. 输入学生学号和三门课的成绩，求出每位学生的平均分并按学生成绩优劣打印一张成绩单.

第八章　字 符 运 算

FORTRAN77 对 FORTRAN66 的重要改进之一就是增加了“字符运算”功能. 它允许使用字符型数据(包括字符常数,字符型变量,字符数组,字符型表达式),允许对字符数据进行运算和处理(例如用字符赋值语句进行字符数据的赋值,输入输出字符数据,取子串,统计字符个数等). 这样就使 FORTRAN 语言的应用范围大为扩展,除了可以用于数值运算之外,还可用于非数值运算,对各种文字材料进行处理.

FORTRAN77 全集和 FORTRAN77 子集包括了不同的字符处理功能。例如全集允许使用“子字符串”,而子集则不允许. 读者应了解所用计算机系统采用的是 FORTRAN 全集还是子集.

§8.1　字符型常数

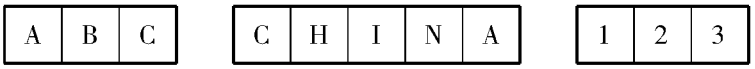
FORTRAN77 规定字符常数必须用两个撇号括起来. 例如:' ABC',' CHINA',' 124','GOOD-BYE','HELLO','WANG LI'等都是字符常数. 注意两边的撇号是字符常数的起止分界符号,不是字符常数的一部分. 字符常数中的空格也是一个有效字符. 例如:'AB CD'这个字符常数中共有五个字符:A,B,空格,C,D.

如果希望字符常数中包括撇号,该撇号应该用两个连续的撇号代表. 例如:'IT S A BOOK',它包括 11 个字符:IT'S A BOOK.

字符常数又称字符串,由计算机系统所允许使用的字符组成. 字符常数的长度就是字符常数中字符的个数. 例如,字符常数'BEIJING'的长度为 7.

注意区别字符常数和数值常数的不同性质.'123'是字符常数,它只是三个字符(1,2,3)的顺序排列,并不代表数值 123,不能用来进行数值运算,例如:3.6 ＊'123'是错误的.

在内存中一个字符占一个字节. 如:



FORTRAN66　允许使用 Hollerith 常数,例如:3 HABC 代表'ABC',有三个字符(A,B,C). 可以用于赋值语句和 DATA 语句. 如:

```
C = 5HABCDE
```

将 ABCDE 五个字符赋给字符型变量 C. FORTRAN77 则取消了这种形式的字符常数,请读者注意. 如果用 FORTRAN77 的编译系统编译用 FORTRAN66 写的源程序,必须事先加以修改(但有的计算机所用的 FORTRAN77 编译系统仍然允许使用 Hollerith 常数).

§8.2　字符型变量和字符型数组

字符型变量和字符型数组必须事先用类型语句或 IMPLICIT 语句加以定义(或称“说明”).

用来定义(说明)字符型变量或数组的类型语句为 CHARACTER 语句. 其一般形式为:

CHARACTER[*长度[,]] 变量或数组表

例如:

```
CHARACTER *5 A,X(6),Y(4,3)
```

它定义了 A 为字符型变量,长度为 5(5 个字符). X 为字符型一维数组,有 6 个元素,每个元素为 5 个字符长度,Y 为字符型二维数组,它的大小为 4×3 共 12 个元素,每个元素的字符长度也是 5.

一个字符型数组中各元素的长度是相同的,即不允许同一数组中有的元素长度为 5,而另一些元素长度为 6.

可以用一个 CHARACTER 语句来说明不同长度的字符型变量和数组. 如:

```
CHARACTER A*3,B*5,C(3,4)*4
```

它表示,字符型变量 A 长度为 3,B 长度为 5,数组 C(二维数组,大小为 3×4)中的元素的长度为 4.

还可以是以上两种方式的结合:

```
CHARACTER *8  A*6,N(2,6),C*3,D,T(3,3)*4
```

它说明字符型变量 A 长度为 6,C 长度为 3,二维数组 T 的各元素的长度为 4. 其他的(数组 N 和变量 D)长度均为 8.

也就是说,如果在 CHARACTER 语句中,变量(或数组)的后面有一个“*”号,“*”的后面是一个无符号的非零整数,则这个整数就是该变量(数组)的长度. 如果“*”号出现在 CHARACTER 的后面,则“*”后面的整数指出该语句中所有变量(或数组)的长度. 如果二者有矛盾(即“统一说明”和“个别说明”不一致),则“个别说明”优先于“统一说明”.

如果省略“*”号和长度,则隐含长度为 1. 如:

```
CHARACTER A,X,Y
```

A,X,Y 均是长度为 1 的字符型变量.

也可以用 IMPLICIT 语句说明,如:

```
IMPLICIT CHARACTER*5 (A,X,T)
```

指定了凡以字母 A,X,T 开头的变量或数组,均为字符型,长度为 5.

可以用 DIMENSION 语句和 CHARACTER 语句来定义字符数组:

```
DIMENSION A(3,5)
CHARACTER*5 A,B
```

数组 A(大小为 3×5)为字符型,长度为 5. 同时定义了字符变量 B,长度亦为 5.

除了可以对变量或数组定义其类型之外,还可以对符号常数定义类型,如:

```
CHARACTER*20 NAME
PARAMETER (NAME='FORTRAN 77 LANGUAGE')
```

先定义 NAME 为字符型,长度为 20,再用 PARAMETER 语句定义它为符号常数,并使其代表

'FORTRAN 77 LANGUAGE␣'二十个字符.

FORTRAN77 全集允许在定义字符型符号常数时不具体指出字符长度，而用(*)表示它可以是任意的长度.例如：

```
CHARACTER NAME *(*)  (或 CHARACTER *(*) NAME)
PARAMETER (NAME = 'BEIJING')
```

系统会根据 PARAMETER 语句自动将 NAME 的长度定为 7.这对于使用字符型符号常数是很方便的，不必去数字符的个数，而且便于修改符号常数的内容，例如，想使 NAME 代表'SHANGHAI'，只需修改参数语句：

```
PARACTER (NAME = 'SHANGHAI')
```

而不必修改 CHARACTER 语句.

用 CHARACTER 语句定义字符型符号常数时，CHARACTER 语句应在 PARAMETER 语句之前.

CHARACTER 语句中的长度可以是整数，在 FORTRAN77 全集中还可以是整型表达式.如：

```
CHARACTER A*(2+6),B*(3*7)
```

A 的长度为 8，B 的长度为 21.也可以用：

```
PARAMETER (I=3,J=6)
CHARACTER A*(2+J),B*(I*7)
```

(注意：A 和 B 不是符号常数，所以 CHARACTER 语句不必在 PARAMETER 语句之前.相反，要先用 PARAMETER 语句定义 I 和 J 的值).

§8.3 字符变量的赋值

字符变量和字符数组用于(而且只能用于)存放字符型常数.如果已用 CHARACTER 语句定义了变量 A 和数组 B，则

```
CHARACTER *5 A,B(3,4)

A = 'CHINA'
B(2,3) = 'JAPAN'
B(3,1) = 'INDIA'
```

是合法的，而下面的赋值是非法的：

```
B(1,1) = 876.5      (876.5 是数值常数)
B(2,2) = C          (C 不是字符变量)
```

用来对字符型变量或数组元素进行赋值的语句，称为字符赋值语句.算术赋值语句允许赋值号两侧类型不同(整型、实型、双精度型、复型)，在赋值时自动转换类型，而字符赋值语句必须要求赋值号两侧均为字符型，否则出错.

可以用 DATA 语句给字符型变量或数组赋以初值.

```
CHARACTER*3,A,B,C
```

```
DATA A/'NEW'/,B/'OLD'/,C/'YES'
```

在用字符赋值语句和 DATA 语句给字符型变量(或数组元素)赋值时,如果"赋予的"和"被赋予的"二者长度不一致,按下面方法处理:

(1) 如果字符变量的长度大于字符常数的长度,则右边补足空格. 如:

```
CHARACTER *6 A
A = 'CHINA'
```

则 A 的六个字节中存放'CHINA'.

C	H	I	N	A	

(2) 如果字符变量的长度小于字符常数的长度,则将字符常数最右端的多余字符截去. 如.

```
A = 'SHANGHAI'
```

SHANGHAI 中最右端的二个字符 AI 被截去,变量 A 的内容为:

S	H	A	N	G	H

用 DATA 语句时,FORTRAN77 全集按上面原则处理,而 FORTRAN77 子集则不允许字符常数长度超过字符变量的长度. 例如:

```
DATA A/'SHANGHAI'/
```

被认为是非法的.

§8.4 字符表达式

字符表达式是用字符运算符把字符常数、字符变量(或字符数组元素)连接起来的式子.

字符运算符只有一种,就是字符连接运算符"//". 它的作用是将两个字符数据连接起来. 它的长度为二者之和. 如:

```
CHARACTER A*7,B*2,C*10
A = 'FORTRAN'
B = '77'
C = A//' '//B
PRINT *,C
END
```

C 的内容为'FORTRAN ␣ 77',共十个字符.

最简单的字符表达式形式是:字符常数、字符变量、字符数组元素、字符子串(见 §8.7)和字符函数(见 §8.8).

字符表达式的值是字符常数.

§8.5 字符关系表达式

字符型数据可以用关系运算符进行比较. 例如:

```
CHARACTER A*3,B*4
```

```
A = 'BOY'
B = 'GIRL'
IF (A. GT. B) THEN
   PRINT *, A
ELSE
   PRINT *, B
END IF
END
```

怎样进行字符数据的比较呢？数值型数据的比较是按照它们的代数值进行比较的. 字符数据的比较是按照字符的代码来进行比较的. 每个字符用一个代码表示. 常用的有两种代码：

（1）ASCII 代码（American Standard Code for Information Interchange，美国信息交换标准代码）. 大多数计算机采用 ASCII 代码. 在 ASCII 码中，字母“A”以二进制码 01 000 001 代表. 按相应的 ASCII 码的大小，字符顺序为：

空格　# $ % & ( ) * + , - . /

0 1 2 3 4 5 6 7 8 9 : ; < = > ? @

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

（2）EBCDIC 代码（Extended Binary-Coded-Decimal Interchange Code）. 扩展的二-十进制交换码）. 字母“A”以 11000001 表示. 按相应的代码大小，字符顺序为：

空格 . < (+ & $ *) ; / , % - > ? : # @ =

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

美国的 IBM 系列机（除 PC 机）和日本 M 系列机多采用 EBCDIC 码.

不论用哪一种代码，共同的规律是：（1）字母 Z 大，A 小；（2）数字 9 大，0 小.（3）空格最小.

字母'A'和数字'1'谁大？在 ASCII 码中'A' > '1'，而在 EBCDIC 码中，'1' > 'A'. 其他专用字符在这两种代码中排列顺序不相同.

附录 IV 中列出了全部字符的 ASCII 代码和 EBCDIC 代码表. 字符的比较大小就是以该表中的代码为基础的.

（1）两个字符相比较，以代码大者为大. 如

'A' < 'F', '0' < '8'.

（2）两个字符表达式比较，将它们的值（是字符常数）逐个字符（由左到右）进行比较. 如果二者全部字符相同，则认为这两个字符表达式相等. 如果二者不相等，则以第一个出现的不同的字符的比较结果为准. 如：

'THEN'　和　'THAT'

前二个字符相同，第三个字符'E' > 'A'. 故认为'THEN' > 'THAT'. 而不问其后的字符比较结果如何（第四个字符'T' > 'N'，但它对比较结果不起作用）.

（3）如果两个字符表达式的字符长度不同，则短的补以空格，直到二者长度相同. 如：

'THE'　和　'THESE'

先将'THE'补空格成'THE␣␣',然后和'THESE'比较. 由于空格最“小”,因此,'THE' < 'THESE'.

(4) 字符常数中的空格也参加比较. 如:

'FORTRAN␣77'不等于'FORTRAN77'

因为前者当中有一空格.

如果参加比较的字符常数都是由字母和空格组成的,有一简便的方法很快可以判断出它们的大小. 这就是按英文字典中字母的顺序. 字典中后面的字比前面的字“大”. 例如:'THEN',字典中在'THAT'之后,因此,'THEN' > 'THAT'. 同样,“BOY”在字典中的位置在“GIRL”之前,因此'BOY' < 'GIRL',故('BOY'. GT. 'GIRL')的值为“假”. 这样就不必去查 ASCII 码了,只要按人们习惯的字典顺序就能决定字符数据比较的结果. 事实上,许多需要处理的字符常数就是由字母组成的(如人名、国家名、商品名等).

如果字符常数中既包含字母又包含数字或其他专用字符(如#$,.( )等),则比较的结果与计算机系统采用哪一种代码(ASCII 代码还是 EBCDIC 代码)有关. 例如:

'FORTRAN'. GT. '3DG4'

当用 ASCII 码时上述关系表达式的值为“真”. 如果用 EBCDIC 码,则关系表达式的值为“假”.

§8.6 字符型数据的输入输出

字符型数据的输入和输出同样可以有表控格式(自由格式)和有格式的输入输出两种类型.

8.6.1 表控格式的输入输出

1. 表控输入

用以下形式的语句:

```
CHARACTER *3 A
READ(*,*) A (或 READ *,A)
```

执行 READ 语句时,输入字符常数给 A. 输入时要用撇号将字符括起来. 如:

'YES' 或 'NO␣'

如果输入字符常数的长度大于字符变量的长度,则右边多余字符被截去. 如输入:

'␣␣YES'

则只取左边三个字符'␣␣Y'输入给 A. 如果输入的字符常数的长度小于字符变量的长度,则将字符常数的右边补空格,使之与字符变量的长度相等. 例如输入:

'NO'

则将它补空格成为'NO␣',然后将这三个字符输入给 A.

用表控输入时应注意:

(1) 如果用一个 READ 语句读入多个数据,在输入的两个数据之间应以空格式逗号相隔. 可以用一个 READ 语句同时输入字符型数据和数值型数据.

(2) 如果在一个输入的记录中提供的数据个数不够(一个记录中数据的个数少于 READ 语

句中变量的个数),则接着从下一个记录中读数据,直到满足所需的数据为止.

(3) 如果一个记录中的数据个数多于 READ 语句中变量的个数,则多余的数据不被读入.

如果已经将 A,B,C,D 定义为长度为 5 的字符变量,有以下 READ 语句:

```
READ(*,*) A,B,C,D
```

可以用以下几种形式之一输入数据:

(i) 'CHINA' 'JAPAN'
'KORIA','INDIA'

(ii) 'CHINA'
'JAPAN'
'KORIA','INDIA'

(iii) 'CHINA','JAPAN','KORIA','INDIA'

2. 表控输出

如果将上述的字符变量 A,B,C,D 用表控格式输出:

```
WRITE(*,*) A,B,C,D
```

则将 A,B,C,D 的值(即字符常数)打印出来,每两个字符数据间空一个或几个空格(不同系统的规定不同). 形式如下:

```
CHINA JAPAN KORIA INDIA
```

请注意:打印出来的字符常数不包括撇号.

8.6.2 格式输入输出

用编辑符 A 或 Aw 来指定输入输出格式.

1. 格式输入

(1) 用 Aw 编辑描述符

"A"是字符型的编辑符,w 指出字段宽度. 例如:

```
      CHARACTER *5 A,B,C*4,D*3
      READ(*,100) A,B,C,D
100   FORMAT(A5,A5,A4,A3)
```

输入数据如下:

```
CHANG ZHANG WANG TAN
 5列   5列   4列  3列
  ↓     ↓     ↓    ↓
  A     B     C    D
```

注意:输入的数据间不必(也不能)用逗号或空格相隔,也不必加撇号. 读入时会自动截取前 5 个字符给变量 A,第 6 到第 10 列的字符给变量 B,11 到 14 列字符给变量 C,15 到 17 列字符给变量 D.

如果 FORMAT 语句改为:

```
100   FORMAT(A5,2X,A5,2X,A4,2X,A3)
```

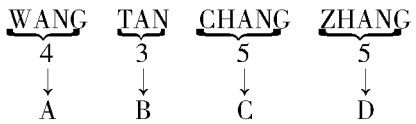
则输入上述数据时应按以下形式:

CHANG␣␣ZHANG␣␣WANG␣␣TAN

在以上例子中，FORMAT 语句中编辑描述符 Aw 中的 w 值恰好等于字符变量的长度（分别为 5,5,4,3）. 如果它们之间长度不相等，会出现什么情况呢？例如：

```
      CHARACTER *5,A,B,C*4,D*3
      READ(*,100) A,B,C,D
100   FORMAT(A4,A3,A5,A5)
```

输入的数据如果是：

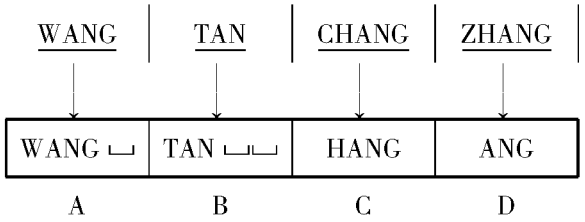


仍按 Aw 的规定截取 w 个字符送给相应的变量. 但在送入时按以下原则处理（设字符变量的定义长度为 l）.

（i）$w=l$ 时，将 w 个字符全部送入变量中.

（ii）$w<l$ 时，在 w 个字符后面补（$l-w$）个空格，然后送给变量. 例如，本例中送给 A 的是'WANG␣'五个字符，送给 B 的是'TAN␣␣'五个字符.

（iii）$w>l$ 时，只取最右边 l 个字符送给变量（注意，不是最左边的 l 个字符，这是和表控输入时不同的，表控输入时是取最左面的 l 个字符）. 可以理解为：字符从右向左一个一个地挤进去，如果 $w>l$，就把最左边的（$w-l$）个字符挤出变量的存储单元. 本例的读入结果如下：



归纳起来，读字符数据的步骤是：① 先按 Aw 中的 w 值截取字符数据；② 把它们变成与字符变量相同的长度；③ 送到相应的变量中.

（2）用 A 编辑符

可以用简单的方式进行格式输入，即不用 Aw 形式而只用一个“A”符号即可. 例如可把上例中的 FORMAT 语句改为：

```
100   FORMAT(A,A,A,A)
```

或

```
100   FORMAT(4A)
```

注意在“A”后面没有 w. 系统会按字符变量的长度来截取各字符常数. 由于变量 A 的长度为 5，因此自动取前 5 个字符送给变量 A，其他相似. 它相当于：

```
100   FORMAT(A5,A5,A4,A3)
```

用这种简单的编辑描述符方便好记，能自动保证所需的字符个数，不易出错. 在一般的情况下，建议用这种通用的形式. 如果字符变量的长度变化，例如：

```
      CHARACTER *10 A,B,C,D
```

而 FORMAT 语句仍然可以不必变化.

2. 格式输出

也是用 A 和 Aw 编辑描述符.

(1) 用 Aw 编辑描述符

```
      WRITE(*,120) A,B,C,D
120   FORMAT(1X,A5,A5,A4,A3)
```

假若 A,B,C,D 中已分别存入'CHANG','ZHANG','WANG','TAN',则打印结果为:

```
CHANGZHANGWANGTAN
```

如果想插入空格,可修改 FORMAT 语句:

```
120   FORMAT(1X,A5,2X,A5,2X,A4,2X,A3)
```

如果格式说明中 Aw 中的 w 不等于字符变量的长度 l,则按以下规则处理:

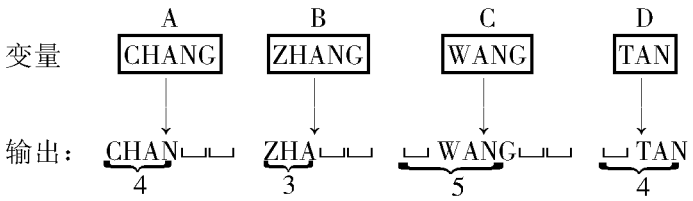
(i) $w < l$,输出字符变量中最左边的 w 个字符.

(ii) $w > l$,在字符变量的值的左边补充$(w - l)$个空格. 即:字符向右对齐,左面补空格.

例如,如将上述 FORMAT 语句改为:

```
120   FORMAT(1X,A4,2X,A3,2X,A5,2X,A4)
```

输出的情况如下:



可以看到,如果不能恰当地选择 Aw 编辑描述符,会造成字符变量的值在输出时被截断的情况. 遇到这种情况时,应很快地想到这是格式说明和变量的长度不匹配的缘故.

(2) 用 A 编辑描述符

按字符变量的长度输出字符. 例如:

```
      CHARACTER A*4,B*3
      A='AAAA'
      B='BBB'
      WRITE(*,10) A,B
10    FORMAT(1X,A,A)
      END
```

输出结果为:

```
AAAABBB
```

自动将 A 的输出字段宽度定为 4(按 A4),B 的输出字段宽度定为 3(按 A3).

如果用以下语句输出,输出的第一个字符被“吃掉”.

```
      WRITE(*,'(A)')A,B
```

输出结果为两行(重复使用格式说明):

AAA

BB

§8.7 子字符串

字符常数又称字符串(String).一个字符串中的一部分(字符串中相邻的几个字符)称为子字符串或字符子串(Substring).子字符串的形式是:在一个字符变量(或字符型数组元素)名字的后面有一对括号,括号内写两个整数(或整型表达式),两者之间用冒号分隔.例如,A(1: 7),B(1)(5: 6)等.

```
CHARACTER * 20 A,B(10) * 10
A = 'FORTRAN ␣ 77 ␣ LANGUAGE ␣'
B(1) = 'MR. ␣ LI ␣␣␣␣'

WRITE ( * , * ) A (1: 7),B(1) (5: 6)
```

在 WRITE 语句中的 A(1: 7)和 B(1)(5: 6)就是子字符串.括号中两个整数代表子字符串在字符变量(或数组元素)中的起止位置.A(1: 7)表示此子字符串是字符变量 A 中左面第一个字符到第七个字符.B(1)(5: 6)表示它是字符型数组元素 B(1)中左边开始的第五、第六两个字符.将这两个子字符串打印出来,结果是:

```
FORTRANLI
```

由此可见,利用子字符串可以从一个字符变量或数组元素中选取所需的一部分.在程序中子字符串可以单独使用.例如,可以直接打印子字符串的值,也可以用来与其他字符数据进行比较,还可以用子字符串组成字符表达式.例如:

$$A(1: 8)//A(12: 19)$$

的值是'FORTRAN ␣ LANGUAGE',可以将它赋给另一个字符变量 C.在字符处理中,子字符串是十分有用的,利用它可以从已有的字符常数中任选其中一些部分组成新的字符常数.

子字符串的一般形式可以表示为:

字符变量名(数组元素名)($\mathbf{e_1}$: $\mathbf{e_2}$)

e_1 和 e_2 是整型表达式.也可以不写 e_1 或 e_2,如果不出现 e_1,则表示 $e_1 = 1$.如不写 e_2,则隐含 e_2 的值为字符变量的长度 l.冒号不能省略.如果 e_1 和 e_2 都省略,隐含 $e_1 = 1$ 和 $e_2 = l$.

下页表 8.1 是当 A = 'FORTRAN ␣ 77 ␣ LANGUAGE ␣'时各子字符串的值.

子字符串中的(e_1: e_2)称为子串表达式,它决定子串在字符变量中的起止位置.应当满足以下关系:

$$1 \leqslant e_1 \leqslant e_2 \leqslant l$$

子字符串的长度为:$e_2 - e_1 + 1$

可以将子字符串的值赋予一个字符型变量或数组元素.如:C = A(1: 7)

子字符串	子字符串的值	说　明

A(9: 10)	77	
A(13:)	LANGUAGE ␣	相当于 A(13: 20)
A(: 7)	FORTRAN	相当于 A(1: 7)
A(1: 1)	F	
A(:)	FORTRAN ␣ 77 ␣ LANGUAGE ␣	相当于 A
A(1: 21)	出错	21 > l

其中 C 是已定义的字符变量，也可以将一个子字符串的值赋给另一个子字符串，如：

```
C(1: 12) = A(9: 20)
A(8: 14) = A(1: 7)
```

但赋值号的两侧不能引用同一个子字符串. 即，同一个字符变量中的同一个位置的字符不应出现在赋值号两侧. 如：

```
A(5: 9) = A(8: 12)
```

是非法的，因为，A(5: 9)和 A(8: 12)都包含了同一个子字符串 A(8: 9). 应该改为：

```
T = A(8: 12)
A(5: 9) = T
```

但也有的 FORTRAN77 编译系统允许这种用法。

FORTRAN77 子集不能使用子字符串.

§8.8 用于字符处理的内部函数

在字符处理中常要用到一些内部函数：

(1) 求字符串长度的函数 LEN(C)

有时不能事先确切知道字符串的长度，例如，用(*)代表字符长度时：

```
CHARACTER *( * ) A
PARAMETER ( A = 'CHINA' )
```

可以不必数 A 中字符个数，而用 LEN(A)求出 A 的字符长度，它的值为 5.

(2)“字符序号转换为字符”函数 CHAR(I)

其中 I 为整型表达式. 在 ASCII 代码或 EBCDIC 码中每个字符有一相应的顺序号码，从 0 开始. 如果一共有 n 个字符，则号码从 0 到 n - 1. 例如，在 ASCII 码中，字符与号码关系如下：

号码	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
字符	空格	$	,	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	=	A	B	C	D	E

号码	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
字符	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

如果有函数 CHAR(25),它的值就是字符'C'. CHAR(16)的值是字符'5'.

如果某计算机系统采用 EBCDIC 码,则应按它的字符整理序列中顺序号与字符对应关系去求字符.

(3)"转换为字符序号"函数　ICHAR(C)

是 CHAR 函数的反函数. 它的作用是将一个字符转换成与它相应的字符整理序列中的序号. 例如:按上表,ICHAR('A')的值为 23,ICHAR('2')的值为 13.

有时在程序中为方便起见,将一些算术量也作为字符数据一起输入,然后在程序中再转换为数值. 例如,输入'2468',希望转换成数 2468. 怎样才能得到这一转换关系呢? 从上面知道:ICHAR('2')的值为 13 而不是 2,要得到 2 就应减去 11(11 是'0'的序号),即 ICHAR('2') - ICHAR('0'),它的值就等于 2 了. 要想将'2468'转换成 2468,可以用下面一组语句:

```
      A = '2468'
      N = 0
      M = ICHAR('0')      (得到 M 的值为 11)
      DO 1, I = 1, 4
      N = N * 10 + ICHAR(A(I: I)) - M
1     CONTINUE
```

最后得到的 N 就是 2468.

(4)"子串位置"函数　INDEX(a_1, a_2)

a_1 和 a_2 是两个字符串. 如果 a_2 是 a_1 的一个子串,则 INDEX 函数的值为 a_2 在 a_1 中第一次出现的起始位置,如果 a_2 不是 a_1 的子串,则函数值为 0. 若有 A = '␣␣WANG␣LI', B = 'WANG',则

```
INDEX (A, B)
```

的值为 3. 因为 WANG 是 A 的子串,是从 A 中第三个字符位置开始的.

可以利用这个函数来判断一个字符串中是否包含另一个字符串. 如果包含,则 INDEX 函数值 >0,否则 =0.

(5)"字符大于或等于"函数 LGE(a_1, a_2).

a_1 和 a_2 是字符型变量,按 ASCII 码将 a_1 和 a_2 比较,如果 $a_1 \geqslant a_2$,则函数值为真,否则为假.

(6)"字符大于"函数　LGT(a_1, a_2)

按上述规定,若 $a_1 > a_2$,则函数值为真,否则为假.

(7)"字符小于或等于"函数　LLE(a_1, a_2)

若 $a_1 \leqslant a_2$,函数值为真,否则为假.

(8)"字符小于"函数　LLT(a_1, a_2).

若 $a_1 < a_2$，函数值为真，否则为假.

FORTRAN 77 全集提供以上函数，而 FORTRAN 77 子集不包括这些函数.

§8.9 程 序 举 例

例 **1** 输入十个国家名，每个的长度不超过 20 个字符. 要求按英文字母顺序把它们打印出来.

这是一个字符数据排序问题，其方法与数值数据排序方法相同，今采用“选择法”排序，即将 10 个数据放在数组 A 中. 将 A(1)与 A(2)到 A(10)比较，每次比较后把值“小”的那个数组元素的下标记下来，在全部比较完之后将“最小的”元素值与 A(1)对换. 这样要比每比较一次就对换一次(如果 A(1)“大”于与它比较的元素值的话)效率高一些.

程序如下：

```
      CHARACTER*20 A(10),TEMP
      READ(*,100) N,A
      DO 1,I=1,N-1
        K=I
        DO 2,J=I+1,N
          IF (A(K).GT. A(J)) K=J
2       CONTINUE
        TEMP=A (I)
        A (I)=A (K)
        A (K)=TEMP
1     CONTINUE
      WRITE(*,200) A
100   FORMAT (I3/ 4(A20))
200   FORMAT (1X,4A20)
      END
```

请注意标号为 100 的 FORMAT 语句. 第一行先输入国家数(今为 10)，第二行开始输入各国家名. 每行内输入四个国家名. 输入的数据如下(注意每个国家名必须占 20 列宽度)：

```
10
CHINA               JAPAN               INDIA               HOLLAND
ENGLAND             CANADA              KORIA               AMERICAN
KUBA                POLAND
```

输出结果如下(已按顺序排列好)：

```
AMERICAN            CANADA              CHINA               ENGLAND
HOLLAND             INDIA               JAPAN               KORIA
KUBA                POLAND
```

例 **2** 有若干张卡片，上记“RED”(红)，“BLUE”(蓝)，“GREEN”(绿)，要求输入计算机后能统计出红、蓝、绿卡片各有多少张. 如果输入的字符不是“RED”，“BLUE”，“GREEN”，应打印出“第几张卡片上有错”的信息.

程序编写如下：

```
      DIMENSION COLOR(4),KOUNT(3)
      CHARACTER COLOR*5,COLORI*5
      DATA COLOR/'RED','BLUE','GREEN','
      KUN=0
      DO 1,I=1,3
        KOUNT (I)=0
1     CONTINUE
      WRITE (*,1000)
20    READ (5,1001) COLORI
      IF (COLORI. NE. COLOR(4)) THEN
        KNT=KNT+1
        DO 30,I=1,3
          IF(COLORI. EQ. COLOR (I)) KOUNT (I)=KOUNT (I)+1
30      CONTINUE
        IF (COLORI. NE. COLOR(1). AND.
     $      COLORI. NE. COLOR(2). AND.
     $      COLORI. NE. COLOR(3))
     $        WRITE (*,1002) COLORI,KNT
        GO TO 20
      END IF
1000  FORMAT('ENTRIES ON FOLLOWING CARDS INCORRECT'/
     $       'CARD COLUMNS: 12345')
1001  'FORMAT (A5)
1002  FORMAT (' ',A5,' ON CARD NUMBER',I3)
1003  'FORMAT (/' TOTAL NUMBER OF CARDS READ=',I3/
     $        'OF WHICH',I4,' ARE RED'/1X,I12,' ARE BLUE'/
     $    1X,I12,'ARE GREEN')
      WRITE (*,1003) KNT,(KOUNT (I),I=1,3)
      END
```

程序开始运行后先打印出：

```
ENTRIES ON FOLLOWING CARDS INCORRECT
CARD COLUMNS: 12345
```

从读卡机上输入一组卡片：

```
RED
RED
BLUE
GREEN
 RED
BLVE
```

```
 BLUE
GREN
GREEN
RED
BLUE
RED
```

打印出以下信息通知你第5,6,7,8张卡片上的数据有错.

```
 RED   ON CARD NUMBER 5 (“R”不在第一列)
BLVE   ON CARD NUMBER 6 (“U”错为“V”)
 BLUE  ON CARD NUMBER 7 (“B”不在第一列)
GREN   ON CARD NUMBER 8 (GREEN 拼错)

TOTAL NUMBER OF CARDS READ = 12
OF WHICH 4 ARE RED
         2 ARE BLUE
         2 ARE GREEN
```

请注意:

(1) 第五张卡片上第一列空白,成了“␣RED”,在程序中被认为不是“RED”,不合法,故认为数据有错. 第七张卡片类似. 第6,8张拼写有错.

(2) 如果不从卡片输入而从终端键盘输入数据,运行结果是相同的,只是输入一个“错误”的记录后,立即输出一个“错误通知”.

(3) 请仔细分析FORMAT语句的格式说明,以及对输出格式的影响.

例**3** 有一篇文章,包括若干行,每行有40个字符. 要求统计出每一行中包含字母“A”的个数. 并要求将原文重新打印出来并编上行号.

解**1** 将每一行40个字符放在一个数组J中,每个数组元素存放一个字符. 在每一行中,逐个地将J(1)到J(40)与字符“A”比较,如果相等,就使NUMBA累加1. 程序如下:

```
      CHARACTER * 1 J(40),K
      DATA K/'A'/
      LINE = 0
      READ (5, * ) N
10    IF (LINE . LT. N) THEN
            READ (5,1000) J
            LINE = LINE + 1
            NUMBA = 0
            DO 20,L = 1,40
                IF (J(L) . EQ. K) NUMBA = NUMBA + 1
20          CONTINUE
            WRITE ( * ,1001) LINE,J,NUMBA
            GOTO 10
```

```
      END IF
1000  FORMAT(40A1)
1001  FORMAT('0',I3,3X,40A1,'NUMBER A S = ',I3)
      END
```

如果用读卡机输入数据，第一张卡片为总行数（今为 5 行文字）. 从第二张卡片开始是五行文字. 设输入数据如下：

```
5
THIS IS THE DATA FOR PROGRAM ALPHA
WHICH NUMBERS THE LINES,OUTPUTS THE
LINES,AND COUNTS THE NUMBER OF TIMES
THAT THE LETTER A OCCURS IN EACH LINE
OF INPUT DATA.
```

共输入六张卡片. 输出结果如下：

```
1 THIS IS THE DATA FOR PROGRAM ALPHA          NUMBER A'S =  5
2 WHICH NUMBERS THE LINES,OUTPUTS THE         NUMBER A'S =  0
3 LINE,AND COUNTS THE NUMBER OF TIMES         NUMBER A'S =  1
4 THAT THE LETTER A OCCURS IN EACH LINE       NUMBER A'S =  3
5 OF INPUT DATA.                              NUMBER A'S =  2
```

如果想改为统计每行中字母“N”的数目，只需修改 DATA 语句即可. 如果每张卡片上字符数为 80，只需将第 1，9，15，16 行中的 40 全改为 80 即可.

请读者考虑以下情况怎样处理：

（1）除了统计各行中字母“A”的个数外，还要统计全文中字母“A”的个数.

（2）事先不知道有多少行文字，无法提前输入行数 N. 以最后输入一张空白卡片作为“结束标志”，程序应作哪些修改？

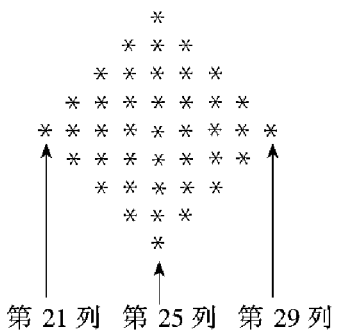
（3）想统计全文和各行中有多少个英文单词，应如何处理.

解 2　不用数组，改用子字符串来处理. 将 J 定义为包含 80 个字符的字符变量，读入一行的字符全存放在 J 中. 每一次取出一个字符（子字符串 J(L: L)代表 J 中第 L 个字符），判断其是否等于“A”，如相等就使 NUMBA 累加 1.

```
      CHARACTER J*80,K*1
      DATA K/'A'/
      LINE=0
      READ(*,*) N
10    IF(LINE.LT. N) THEN
          READ(*,1000) J
          LINE=LINE+1
          NUMBA=0
          DO 20,L=1,40
              IF(J(L: L). EQ. K) NUMBA=NUMBA+1
20        CONTINUE
```

```
            WRITE ( * ,1001) LINE,J,NUMBA
            GOTO 10
        END IF
1000    FORMAT ( A40)
1001    FORMAT (1X,I3,3X,A40,'NUMBER A' 'S = ',I4)
        END
```

例 **4**　打印出以下图案，图案的竖向中心线要求打在程序纸的第 25 列上.



请先考虑遇到这类问题怎样着手？整理出思路.

定义一个字符变量 LINE，长度为 50，先使它为全空格，然后在应该有“ * ”的位置放入一个“ * ”符号. 为打印这个图案，可分解为两个步骤：（1）打印上面五行. 它的规律是每一行的星号个数是上一行个数加 2，而且星号起始位置向左移一位. 例如第一行“ * ”出现在 25 列，第二行的第一个“ * ”出现在 24 列.（2）打印下面四行（即第六至第九行）. 它的规律是：第六行有七个“ * ”，第七行有五个“ * ”，即下一行星号的个数比上一行少 2，而且第一个“ * ”的位置向右移一位.

程序如下：

```
        CHARACTER LINE * 50
        DO 10,I = 1,5                          ┐
            LINE = ' '                         │
            N = 25 - I                         │
            DO 20,L = 1,2 * I - 1              ├ 打印上面 5 行
                J = N + L                      ┐ 存入“ * ”
                LINE( J: J) = ' * '            ┘ │
20          CONTINUE                           ┘
        WRITE( * , * )LINE
10      CONTINUE
        DO 30,I = 1,4                          ┐
            LINE = ' '                         │
            N = 20 + I                         │
            DO 40,L = 1,9 - 2 * I              ├ 打印下面四行
                J = N + L                      ┐ 存入“ * ”
                LINE ( J: J) = ' * '           ┘ │
40          CONTINUE                           │
            WRITE ( * , * ) LINE               ┘
30      CONTINUE
        END
```

处理上面五行的情况如下：

行　数 （I）	N （25－I）	内循环执行次数 （2I－1）	“＊”从第几列开始 （N＋1）	有几个“＊”
1	24	1	25	＊
2	23	3	24	＊＊＊
3	22	5	23	＊＊＊＊＊
4	21	7	22	＊＊＊＊＊＊＊
5	20	9	21	＊＊＊＊＊＊＊＊＊

处理下面四行的情况如下：

行　数 （I）	N （20＋I）	内循环执行次数 （9－2I）	“＊”从第几列开始 （N＋1）	有几个“＊”
1	21	7	22	＊＊＊＊＊＊＊
2	22	5	23	＊＊＊＊＊
3	23	3	24	＊＊＊
4	24	1	25	＊

写这个程序的关键在找出循环次数和行数 I 的关系，以及“＊”起始位置与行数的关系. 例如，第二行时，I＝2，循环次数为（2I－1）＝3，起始位置为 J＝N＋1＝（25－I＋1）＝23（起始位置为 L＝1 时的情况，故 J＝N＋1）.

习　题

1. 希望把'HOW','DO','YOU','DO'分别放入 A,B,C,D 四个字符变量中. 已有：

CHARACTER A＊3,B＊2,C＊3,D＊2

用以下 READ 语句和 FORMAT 语句，问应如何输入数据.

READ（＊,100）A,B,C,D

（1）100 FORMAT（A3,A2,A3,A2）

（2）100 FORMAT（A5,A4,A3,A5）

（3）100 FORMAT（A,A,A,A）

（4）100 FORMAT（1X,A5,2X,A4,2X,A3,2X,A3）

2. 如果已由上题将'HOW','DO','YOU','DO'放入 A,B,C,D 中，按以下 WRITE 语句和 FORMAT 语句输出的结果如何？

WRITE（＊,150）A,B,C,D

（1）150 FORMAT（1X,A3,A2,A3,A2）

（2）150 FORMAT（1X,A5,A4,A5,A4）

（3）150 FORMAT（1X,A3,2X,A2,2X,A3,2X,A2）

（4）150 FORMAT（1X,A2,A3,A2,A3）

（5）150 FORMAT（A3,A3,A4,A3）

（6）150 FORMAT（1X,4A）

3. 按 ASCII 代码字符整理序列，写出下面字符关系表达式的值.

（1）'THE'.GT.'THAT'　　（2）DO'.LE.'DID'

（3）'ABC'.LT.'123'　　（4）'SHANGHAI'.GE.'BEIJING'

（5）'012'.LT.'316'　　（6）'3DG4'.GT.'WHY'

（7）'WHAT'.LE.'####'　　（8）'LEST␣'.EQ.'␣LEST'

4. 有一批图书，每本书有：书名（NAME）、作者（AUTHOR）、编号（NUM）、出版日期（DATE）四个数据. 希望输入后按书名的字母顺序将各书的记录排列好，供以后查询. 今输入一本书的书名，如果查到库中有此书，打印出此书的书名、作者、编号和出版日期. 如果查不到此书则打印出“无此书”.

5. 有一篇英文文章，编程序统计出文章中的英文单词的个数和空格的个数.

6. 在同一个坐标系上打印 sinx 和 cosx 两条曲线，要求打印出轴线，每隔 10°打印一个点，正弦曲线的轨迹用'·'表示，余弦曲线的轨迹用'*'表示.

7. 打印出下面的图案.

```
* * * * *
  * * * * *
    * * * * *
      * * * * *
        * * * * *
```

8. 输入一批国家的名字，要求把国名第一个字母小于'N'的国家名打印出来.

9. 有一篇文章，要求统计出单词'THE'出现的次数. 并指出它在第几行第几个字符位置开始（提示：用 INDEX 函数）.

10. 有一批图书，书名已输入计算机. 如果查询者记不住书的英文全名，而只记得其中一个英文单词，要求把含有此单词的书名全部打印出来.

第九章　语 句 函 数

在前面各章中，我们已经使用了象 SQRT，MOD，SIN 这样一些内部函数，这些内部函数的使用大大地简化了我们的程序，给程序设计提供了很大的方便.

在 FORTRAN 中，一共有三类函数，这就是内部函数、语句函数和外部函数（又称函数子程序）. 内部函数是由 FORTRAN 编译程序提供的，使用者只要会用就可以. 而语句函数和函数子程序都是程序设计者根据自己的需要由自己来设计和定义的. 但无论是哪种函数，都用以下相同的方式引用：

函数名（$\mathbf{a_1}$，$\mathbf{a_2}$，…，$\mathbf{a_n}$）

此处，a_1，a_2，…，a_n 分别代表函数的自变量，在程序中引用函数时把它们称为实在参数（简称实参）. 函数引用的结果只是得到一个函数值，因此，函数引用不能作为一个单独的语句，而只能作为表达式的一部分，它可以出现在任何可以出现表达式的地方（如出现在赋值语句的右边，输出语句的输出表中，等等. 当然也可以在函数调用中作为实在参数）.

内部函数在第三章已作过介绍，读者在引用时可参考附录 II. 本章将介绍语句函数，函数子程序在下一章讨论.

§9.1　语句函数的定义

在程序中有时可能在好几处需要进行同样的某种表达式计算，而这种计算又不是某个内部函数所能完成的，这时，程序设计者可以自己来定义一个语句函数，通过引用语句函数来实现这种特殊的运算.

语句函数必须在需要引用该函数的程序单位内，在一条语句内定义完，因此称为语句函数. 它的定义形式如下：

函数名（$\mathbf{a_1}$，$\mathbf{a_2}$，…，$\mathbf{a_n}$）= 表达式

以上是一条完整的 FORTRAN 语句. 其中，等号右边的表达式是需要求得函数值的那个表达式. 在表达式内可以包含常数、变量、语句函数的参数、数组元素，还可以包含内部函数、语句函数（已在此之前定义过）以及函数子程序的引用. 但不能自己引用自己. 语句函数定义语句必须放在所有可执行语句之前，其他说明语句之后.

在语句函数定义语句中，等号左边的函数名可以是任意合法的 FORTRAN 名字，但不得和同一程序单位内的其他名字相重. 它的类型可以由类型说明语句说明，如果没有用类型说明语句说明，则遵循隐含规则. 函数名的类型并不是无关紧要的，因为它确定了函数值的类型.

在函数名后一对括号中的 a_1，a_2，…，a_n 是函数的自变量，在 FORTRAN 中称它们为虚拟参数，通常简称为虚参. 虚参只能用变量来表示. 它们只是用来在形式上规定函数自变量的个数、顺

序和类型.

以下是一条语句函数定义语句：

```
DA(A,B) = SQRT(B * B + A * A * A)
```

此语句定义了一个实型函数，函数具有两个实型参数，第一个参数将在表达式中乘以三次方，第二个参数将在表达式中乘以二次方，函数 DA 代表这两项之和的平方根值.

如果把定义语句改写成：

```
INTEGER DB
DB(A,B) = SQRT(B * B + A * A * A)
```

则 DB 是一个整型函数，它的值为两项之和平方根值的整数部分.

在引用语句函数时，必须用实参来代替虚参. 实参具有确定的值，它可以是任意合法的算术表达式，但实参必须与对应的虚参类型一致. 可以用以下语句来引用以上两个函数：

```
X1 = DA(2.0,3.0)
X2 = DB(2.0,3.0)
```

这时 X1 得到的是 4.123105，而 X2 得到的是 4.0.

在语句函数定义语句中，等号左边括号中的虚参，其前后次序由定义者任意选定. 但在引用时则必须在含义上与定义的顺序一致. 例如以下引用 DA 函数的语句：

```
X1 = DA(3.0,2.0)
```

赋给 X1 的值将是 5.567764 而不是 4.123105.

在以下的语句中定义了一个整型函数 NUM：

```
NUM(M,N,X) = M * INT(X) + N
```

函数包含三个虚参，其中 M,N 为整型，X 为实型. 所得函数值为整型. 以下 WRITE 语句：

```
WRITE( * , * )NUM(2,4,5.2)
```

输出结果为 14.

如果把以上的函数引用写成：

```
WRITE( * , * )NUM(2,4,5)
```

则产生一个错误. 因为按照语句函数定义语句的规定，第三个虚参应该是实型. 而以上引用中，第三个实参是整型，实参和虚参的类型不一致因而产生错误.

以下的函数引用也是错误的(假定 L 已经有确定的值)：

```
IF(NUM(L,2).GE.K)THEN
```

因为实参的个数少于虚参的个数.

在语句函数引用时，实参和虚参必须在个数、类型、含义上一一对应. 这一要求和内部函数的引用完全相同.

语句函数定义语句中的虚参实际并不占用内存单元，因此并不影响同一程序单位中使用这些名字作为普通的变量，但是同名的虚参和变量类型相同. 例如在以下程序段中：

```
INTEGER C
```

```
      RES(X,Y,C) = X * *C + 1.6 * Y * *C

      DO 2 C = 1,3
        DO 1 I = 1,4
          X = RES(A,B,I)

1       CONTINUE
2     CONTINUE
```

变量 C 和 X 的使用，对语句函数不产生任何影响.

在语句函数定义语句的表达式中，还可以包含并不在虚参中出现的变量. 例如，如果定义了以下函数：

```
FUN(X) = X * *2 + A
```

则以下语句将给 B 赋以 20.5：

```
A = 4.5
B = FUN(4.0)
```

如果在引用 FUN 函数前，A 尚未赋值，则函数的引用将会发生错误.

FORTRAN 77 规定，定义的语句函数可以不带虚参，例如：

```
ST( ) = SQRT(4.586) + EXP(4.586)
```

在引用 ST 函数时，必须带有括号. 如：

```
A = ST( ) * X + Y
```

语句函数同样也可以定义成字符型、逻辑型等类型，并完成各种类型的运算，在此不一一举例.

§9.2 程序举例

例 1 正直角柱体如图 9.1 所示. 已知五组 a，b 和 h，要求分别求出对应的 d.

为了求出 d，必须先求出上底的对角线 c，在程序中可定义求矩形对角线的语句函数 DIAG(X,Y)，五组数分别放在 A，B，H 数组中.

程序如下：

```
      DIMENSION A(5),B(5),H(5),D(5)
      DIAG(X,Y) = SQRT(X * X + Y * Y)
      DO 1 I = 1,5
        READ( * , * ) A(I),B(I),H(I)
1     CONTINUE
      DO 2 I = 1,5
        C = DIAG(A(I),B(I))
        D(I) = DIAG(C,H(I))
2     CONTINUE
```

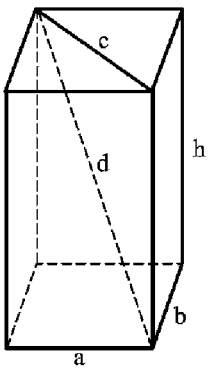


图 9.1

```
      WRITE( * ,100)
100   FORMAT(9X,'A',8X,'B',8X,'H',8X,'D')
      WRITE( * ,110)(A(I),B(I),H(I),D(I),I=1,5)
110   FORMAT(5X,4F9.3)
      END
```

以下是程序一次运行的结果：

A	B	H	D
1.000	2.000	3.000	3.742
4.000	5.000	6.000	8.775
7.000	8.000	9.000	13.928
10.000	11.000	12.000	19.105
13.000	14.000	15.000	24.290

例 **2**　写一个程序，求函数 $\ln(a+\sqrt{1+a^2})$ 在 a 点导数的近似值.

求函数 f 在 a 点的导数可由以下差商公式给出：

$$f'(a)=\lim_{h\to 0}\frac{f(a+h)-f(a-h)}{2h}$$

可使 $h=\frac{1}{2^n}$，n 从 0 变化到 15. 当连续两次求出的两个导数值小于 10^{-5} 时，就认为求得了近似导数值.

程序中可定义两个语句函数，F(A)代表要求导数的函数，FUN(A,H)代表以上差分公式. 求导数点的值放在变量 R 中. 在程序中定义了一个逻辑变量 WORK，当未满足精度要求时，使 WORK 为“真”；当达到精度时，使 WORK 为“假”. 图 9.2 表示了算法框图. 程序如下所示：

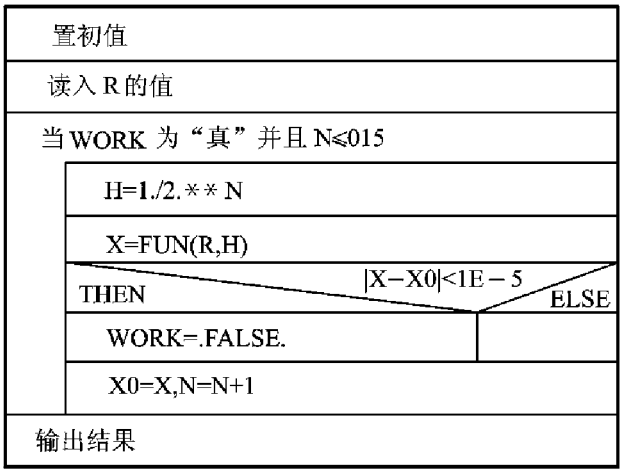


图 9.2

```
      LOGICAL WORK
      F(A)=ALOG(A+SQRT(1+A*A))
      FUN(A,H)=(F(A+H)-F(A-H))/(2.0*H)
      READ( * , * ) R
      WRITE( * , * )'R=',R
      N=0
```

```
      X0 = 0.0
      WORK = .TRUE.
1     IF(WORK. AND. N.LE.15) THEN
        H = 1.0/2.0 * * N
        X = FUN(R,H)
        IF (ABS(X - X0). LT. 1E - 5) WORK = .FALSE.
        X0 = X
        N = N + 1
        GOTO 1
      END IF
      IF (N.LE.15) THEN
        WRITE( * , * )'THE VALUES OF DIFFERENCE QUOTIENT IS: ',X
      ELSE
        WRITE( * , * )'N.GT.15'
      END IF
      END
```

当 R 为 5.5 时，以上程序运行结果得：

```
5.5
R =      5.5000000
THE VALUES OF DIFFERENCE QUOTIENT IS:1.788864E - 001
```

例 **3**　写一个程序求以下方程的一个根（假定方程有根）：

$$x^3 - 5x^2 + 16x - 80 = 0$$

本例中我们采用对分法来求方程的一个根. 现将对分法的算法解释如下：

1. 首先取 X 轴上任意两点 X1，X2，（参考图 9.3），求得 F1 = f(X1)，F2 = f(X2). 若 F1 和 F2 的符号不同，继续下一步；若 F1 和 F2 的符号相同，说明 X1 和 X2 在根的同一侧，因此必须重新再取任意两点，直到 F1 和 F2 的符号不同为止.

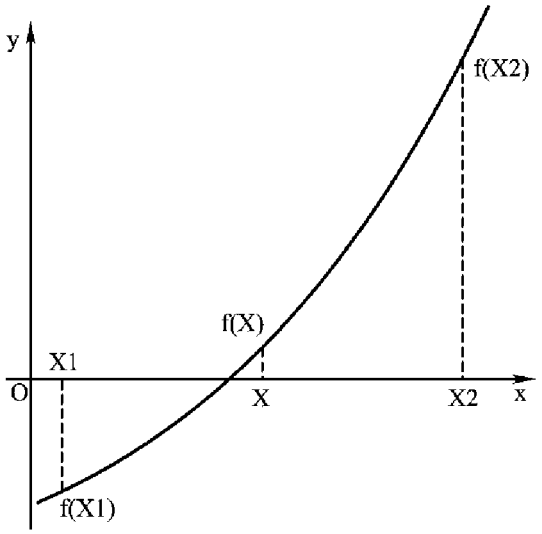


图 9.3

2. 取一新的 X，使 X = (X1 + X2)/2，求出 F = f(X)，若 F 的值与 F1 符号相同，说明曲线与 X 轴的交点在 X 和 X2 之间，因此得新的 X1 = X，F1 = F；反之，则说明交点在 X1 和 X 之间使新的 X2 = X，F2 = F. 重复此过程，当两次所求得的 X 值小于 10^{-5} 时，就认为新的 X 就是方程的根.

图 9.4 中表示了以上算法的框图. 定义一个名为 ROOT 的语句函数来代表求根的函数. 用变量 DELTA 存放前后所求两个根差的绝对值，M 统计重复次数，以避免由于设计中的疏忽而造成无限制重复.

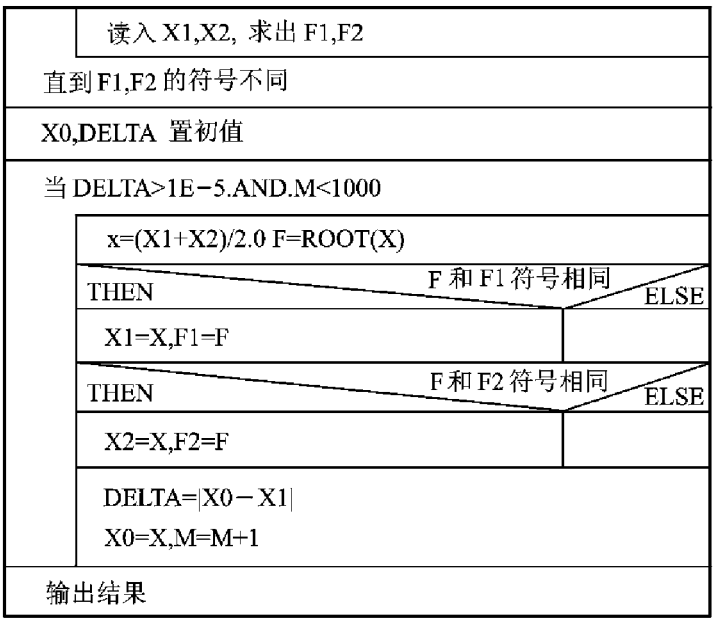


图 9.4

程序如下所示,程序运行结果为 5.000002.

```
      ROOT(X)=((X-5.0)*X+16.0)*X-80.0
10        READ(*,*) X1,X2
          F1=ROOT(X1)
          F2=ROOT(X2)
      IF (SIGN(F1,F2).NE.F1) THEN
      ELSE
      GOTO 10
      END IF
      X0=X1
      DELTA=ABS(X1-X2)
      M=0
20    IF (DELTA.GT.1E-5.AND.M.LT.1000) THEN
          X=(X1+X2)/2.0
          F=ROOT(X)
          IF(SIGN(F,F1).EQ.F)THEN
              X1=X
              F1=F
          END IF
          IF(SIGN(F,F2).EQ.F)THEN
              X2=X
              F2=F
          END IF
          DELTA=ABS(X0-X)
         X0=X
```

```
        M = M + 1
        GOTO 20
    END IF
    IF( M. GT. 1000 )THEN
        WRITE( * , * )'NUM OF ITERATIONS ARE MORE THAN 1000'
    ELSE
        WRITE( * , * )'A ROOT OF THE EQUATION IS: ',X
    END IF
    END
```

习　　题

1. 在程序中定义了以下求本金加利息的语句函数. X 代表本金,N 代表年数,V 代表利率.

$$YIELD(X,N,V) = X*(1.0+V/100.0)**N$$

现有本金 1000 元,利率为 3.5,求两年后本金加利息. 用以下语句引用此语句函数会产生什么结果? 为什么?

```
Z1 = YIELD( 1000,2,3.5 )
Z2 = YIELD( 1000,3.5 )
Z3 = YIELD( 1000.0,2.0,3.5 )
Z4 = YIELD( 1000.0,2 )
```

请写出正确的引用此语句函数的语句.

2. 以下语句函数的功能是什么?

(1)
```
CHARACTER * 10 STRA,STRB,JOIN * 20
JOIN( STRA,STRB ) = STRA // 'V' // STRB
```

(2)
```
PARAMETER( SIZE = 80,STR = 'V' )
CHARACTER * ( SIZE )STRING,STR
LENG( STRING ) = INDEX( STRING // STR,STR ) - 1
```

3. 请写出以下 FNA,FNB 语句函数所定义的多项式的数学公式.

```
POLY( A,B,C,D,X ) = (( A * X + B ) * X + C ) * X + D
FNA( X ) = POLY( 1.0, - 7.8,18.5, - 11.3,X )
FNB( X ) = POLY( 1.0, - 4.0,2.3,5.0,X )
```

4. 人工培养细菌的数目可用以下公式求得:

$$N_0 e^{kt}$$

N_0 是原始细菌数,k 是一个比例常数,t 是时间. 定义一个函数求细菌数,在程序中读入 k,N_0,t(例如,$N_0 = 1000$,$k = 1.5$,$t = 100$),求细菌数.

5. 华氏温度 F 和摄氏温度 C 之间有以下线性关系:

$$F = a \cdot C + b$$

已知水的沸点是 212 华氏温度,100 摄氏温度,而水的熔点是 32 华氏温度,0 摄氏温度,请写出两个语句函数,能使摄氏温度与华氏温度互相转换.

6. 写一个语句函数,把米制的长度转换成英吋(1 英吋≈2.5 cm).

7. 写一个语句函数,用来判断一个实数是否带小数.

8. 写一个语句函数，用来判断一个数是否是偶数.

9. 求方程

$$7x^4 + 6x^3 - 5x^2 + 4x + 3 = 0$$

的一个根（可选用牛顿迭代法）.

10. 已知用梯形法求积分公式如下：

$$\int_a^b f(x)dx = h\left[\frac{1}{2}(f(a) + f(b)) + \sum_{i=1}^{N-1} f(a + ih)\right], \quad h = \frac{b - a}{N}$$

（1）指定 N = 1000，求 $\int_0^{\frac{\pi}{2}} \sqrt{1 + \sin x}\,dx$. 对求积分的函数定义一个语句函数.

（2）N = 2 开始，不断增加 N 值，每次都使 N 加倍（即 2 * N），当所求的两次积分值小于 10^{-5} 时，最后的那一个值就是所求的积分值.

第十章　子　程　序

一个 FORTRAN 程序可以由一个以上程序单位组成，这些程序单位分别称为主程序和子程序. 每个程序单位都可由 FORTRAN 编译程序单独进行编译，进行语法检查，生成目标代码，最后再连接装配在一起，成为一个可执行程序. 图 10.1 是一个可执行 FORTRAN 程序的结构示意图，

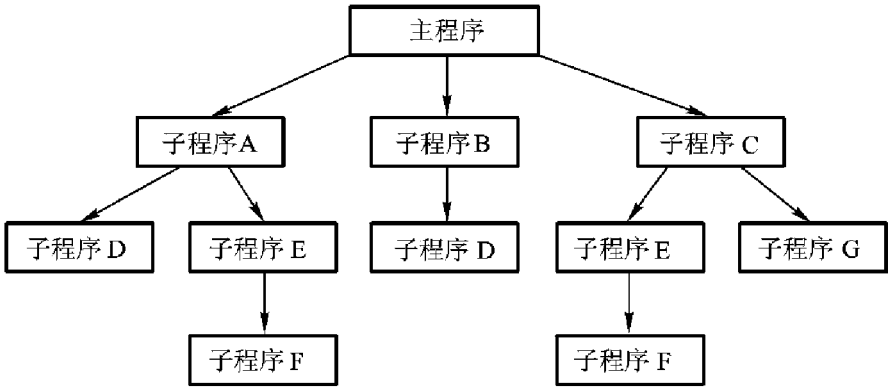


图 10.1

其中每个方框都代表一个可单独编译的程序单位. 可以看到，一个 FORTRAN 程序必须有一个主程序，而且只能有一个主程序. 一个程序中，可以根据需要包含任意数量的子程序，子程序可以被主程序调用，也可以被其他子程序调用，但不能发生直接或间接地自己调用自己. 如图 10.2 中所示的调用关系是错误的.

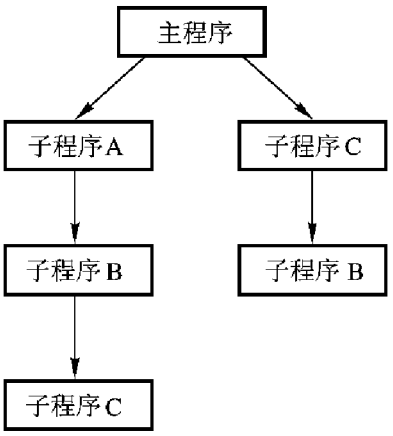


图 10.2

实际上，在本章之前所涉及的都是主程序，这些程序都可由 PROGRAM 语句开头（也可不用 PROGRAM 语句），以 END 语句结束. 在执行程序时，遇到 END 语句，程序就停止运行.

在 FORTRAN 中有三种子程序，即：函数子程序（又称外部函数），子例行程序（又称子例行子程序），数据块子程序，它们的结构如图 10.3 所示. 从形式上看，函数子程序和子例行程序的结构除了第一条语句分别为 FUNCTION 和 SUBROUTINE 语句外，其余几乎和以前讨论的程序没有什么区别. 本章将详细讨论这两种子程序. 数据块子程序虽然也是独立的程序单位，但在其内部只有说明语句，我们将在第十一章中进行介绍.

FUNCTION　函数名(a_1,a_2,…,a_n) 说明语句 可执行语句 END （a）函数子程序结构
SUBROUTINE　子例行程序名(a_1,a_2,…,a_n) 说明语句 可执行语句 END （b）子例行程序结构
BLOCK DATA　名字 说明语句 END （c）数据块子程序结构

图 10.3

§10.1　函数子程序

函数子程序通常称为外部函数，它也是根据程序设计者的需要由自己定义的函数. 和语句函数不同的是，函数子程序是一个由若干语句组成的独立的程序单位. 当某些函数的值不可能用一条语句来求得时，就可借助函数子程序来定义一个函数. 例如，如果在程序中需要引用以下函数：

$$f(x,y,n)=\begin{cases}x+1 & \text{当 } x<y\\ x^n+y^n & \text{当 } x\geqslant y\end{cases}$$

则可以用以下函数子程序来定义这个函数：

```
FUNCTION NF(X,Y,N)
REAL NF
NF = X + 1
IF(X. GE. Y)NF = X * * N + Y * * N
END
```

函数子程序的第一条语句必须是 FUNCTION 语句，它的形式如下：

FUNCTION　函数名($\mathbf{a_1}$,$\mathbf{a_2}$,…,$\mathbf{a_n}$)

函数名遵循 FORTRAN 的取名规则. 函数名的类型决定了函数值的类型. 可以用三种方式来规定函数名的类型. 第一种方式是由 FORTRAN 隐含规则,根据名字中第一个字母来确定函数名是整型或实型. 第二种方式是在函数子程序中,用类型说明语句来说明函数名的类型. 第三种方式是在 FUNCTION 语句中,在关键字 FUNCTION 之前插入一个类型说明. 如上述 NF 函数,若写成:

```
FUNCTION NF(X,Y,N)
NF = NF + 1
IF(X. GE. Y) NF = X * * N + Y * * N
END
```

则 NF 是一个整型函数名,其函数值为整型. 若写成:

```
FUNCTION NF(X,Y,N)
REAL NF
```

则 NF 是一个实型函数名. 上述两条语句可合并成:

```
REAL FUNCTION NF(X,Y,N)
```

同样也说明 NF 是实型函数名. 注意,FUNCTION 前的 REAL 只对函数名 NF 起作用.

在 FUNCTION 语句中,函数名后括号中的 $a_1, a_2, \cdots, a_n$ 称为虚参,它用来代表函数自变量的个数、顺序和类型. 虚参只能用变量名、数组名、虚过程名和 * 号来表示. 本书将不介绍把 * 号作为虚参的有关内容. 关于虚过程名作为虚参的内容将在本章第 4 节中进行介绍. 因此,在此之前,虚参只能是变量名或数组名. 虚参的个数根据需要而定,其类型由函数子程序中的说明语句定义,或者遵循隐含类型规则;数组名必须在函数子程序中用类型说明语句或 DIMENSION 语句定义.

函数子程序的最后一条语句必须是 END 语句. 在函数子程序中,END 语句并不起使程序停止运行的作用. 在 FORTRAN 编译程序对函数子程序进行编译时,由 END 语句通知编译程序,函数子程序作为一个独立的程序单位到此结束. 在执行程序时,若遇到 END 语句,则使函数子程序的执行终止;控制程序执行的控制机构,使调用此函数的那个程序单位继续执行. 通常把这一过程称为把控制返回到调用程序单位.

RETURN 语句只允许出现在函数子程序和子例行程序中,这是一条可执行语句. 当遇到此语句时,就把控制返回到调用程序. 因此,在执行时它与 END 语句有着相同的作用,只是在一个子程序中只能有一条 END 语句,而且必须放在子程序的最后;而 RETURN 语句可以出现在任何需要的地方. 例如上述 NF 函数子程序可以改写成:

```
FUNCTION NF(X,Y,N)
REAL NF
IF(X. LT. Y)THEN
    NF = X + 1
    RETURN
ELSE
```

```
        NF = X * *N + Y * *N
        RETURN
    END IF
    END
```

函数子程序的函数值由函数名带回到调用程序单位，因此在函数子程序中必须把最后所得的函数值赋给函数名. 从形式上说，在函数子程序中，函数名至少必须在赋值语句的赋值号(=)左边或在 READ 语句的输入表项中出现一次. 在上述 NF 函数子程序中，根据需要，NF 在两条赋值语句的赋值号左边出现. 注意：在给函数名赋值时，绝不能把函数名连同括号中的虚参一起出现在赋值语句的等号左边，像下述写法是错误的：

```
NF(X,Y,N) = X + 1
```

在函数子程序中，函数名应该像普通变量那样使用.

除上述指出的以外，在本章之前所介绍的 FORTRAN 语句，都可以在函数子程序中使用. 因为是一个独立的程序单位，因此在函数子程序中除了函数名外，所有名字、语句标号都可以和其他程序单位中的相同，彼此没有关系.

在调用函数子程序的程序单位中，应该对所调用的函数子程序名的类型进行说明，使调用时，不仅与函数子程序名相同，而且类型一致. 例如在调用上述 NF 函数子程序时，必须对 NF 进行说明：

```
REAL NF

W = NF(A,SIN(B),3) + 52.5

END
```

如果不显式说明 NF 为实型，则按隐含规则 NF 被确定为整型. 这样，当执行时，会由于名字的类型不一致而发生错误.

函数子程序的调用方式与内部函数、语句函数的调用方式相同. 调用时的实参必须在个数、类型、顺序上与函数子程序的虚参一一对应，但不要求名字相同. 在执行过程中，实参和虚参按位置一一对应结合. 关于虚实参数的结合将在本章第 3 节中详细叙述.

当程序执行到调用函数子程序的表达式时，控制转向函数子程序，去求函数值；当返回到调用程序时，继续进行此表达式运算或进行函数所在语句中的其他操作.

下面我们来列举一个包含函数子程序的完整程序.

编写程序求：

$$\frac{FUN(A) - FUN(B)}{A - B}$$

FUN 代表下述函数：

$$FUN(x) = \frac{x}{1} + \frac{x^2}{1+2} + \frac{x^3}{1+2+3} + \frac{x^4}{1+2+3+4} + \cdots + \frac{x^n}{1+2+3+\cdots+n}$$

此处$|x|\leqslant 1$，n 的值取决于最后一项是否小于 10^{-5}.

程序的算法可由图 10.4 中的框图表示.

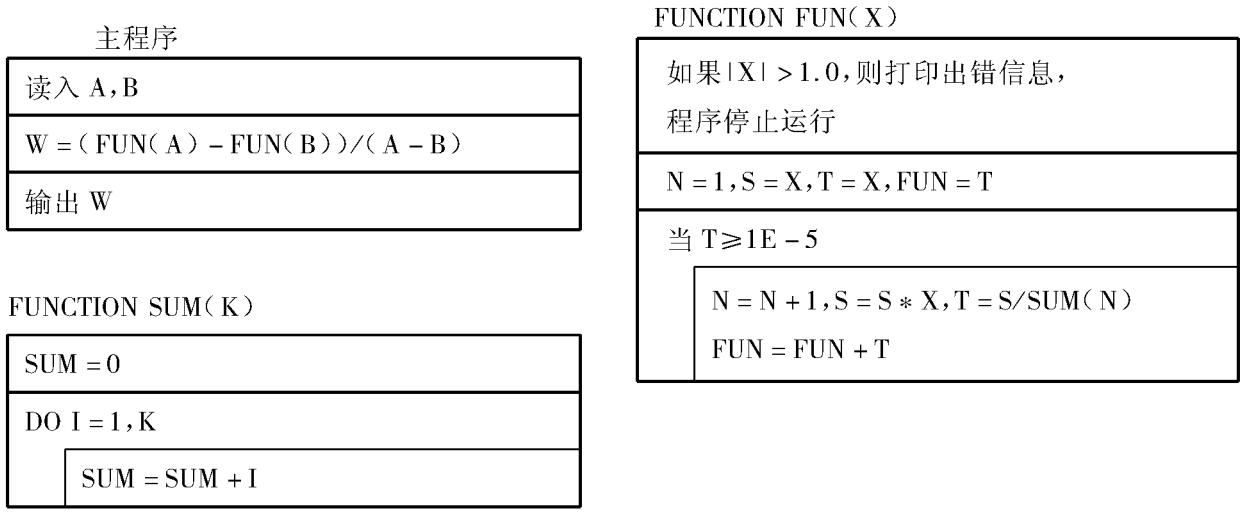


图 10.4

程序清单如下：

```
      PROGRAM MAIN
10    READ(*,*)A,B
      IF(A. EQ. 0. 0)STOP
      W=(FUN(A)-FUN(B))/(A-B)
      WRITE(*,*)A,B,W
      GOTO 10
      END
C
C
      FUNCTION FUN(X)
      INTEGER SUM
      IF(ABS(X). GT. 1.0)THEN
        WRITE(*,*)'ILLEGAL ARGUMENT:',X,'ABS(X)HAS TO LE 1'
        STOP
      END IF
      N=1
      S=X
      T=X
      FUN=X
10    IF(T. GE. 1E-5)THEN
        N=N+1
        S=S*X
        T=S/SUM(N)
        FUN=FUN+T
        GOTO 10
```

```
      END IF
      END
C
C
      FUNCTION SUM(K)
      INTEGER SUM
      SUM = 0
      DO 10 I = 1,K
        SUM = SUM + I
10    CONTINUE
      END
```

当函数名是字符型时，除了必须对函数名进行字符类型说明以外，还必须指定字符串长度. 例如：

```
      FUNCTION CODE(S)
      CHARACTER * 10 CODE
```

也可写成：

```
      CHARACTER * 10 FUNCTION CODE(S)
```

语句规定函数名为字符型，其长度为 10. 当调用此函数子程序时，在调用程序单位中也必须对 CODE 进行字符类型和长度说明，说明的长度应该和函数子程序中说明的一致. 这种说明称为固定长度说明.

在函数子程序中还可以用 * 号来说明函数名的长度，这种说明称为假定长度说明. 例如：

```
      FUNCTION CODE(S)              PROGRAM MAIN
      CHARACTER *( * )CODE          CHARACTER * 15 CODE,COM

                                    IF( CODE( A). GE. COM)THEN
```

以上在主程序中说明了 CODE 函数名的长度为 15，而在 CODE 函数子程序中 CODE 被说明为假定长度. 当主程序调用函数子程序时，这时函数子程序名 CODE 具有长度为 15.

字符函数子程序举例：

输入 50 名学生的学号和数学考试的成绩，要求打印出学生的学号和该学生的得分等级. 等级标准是大于等于 90 分者为 A 级；大于等于 80 分，低于 90 分者为 B 级；大于等于 60 分，低于 80 分者为 C 级；不及格者为 D 级.

根据题意编写一个字符函数子程序 LEVEL(X)，自变量 X 为学生分数，LEVEL(X)得一字符，此字符表示学生得分的等级，视 X 而定.

主程序非常简单，只由两部分组成：

1. 读入 50 个学生的学号和成绩.

2. 打印学生的学号和他们的成绩等级.

在 LEVEL 函数子程序中，A 数组中存放各等级的分数界限，B 数组存放对应的等级字符，当

X 得到一个值时，去判断落在哪个分数段范围内，并使 LEVEL 得相应的字符.

程序如下所示：

```
      PROGRAM MAIN
      DIMENSION NUM(50),S(50)
      CHARACTER LEVEL
      WRITE(*,*)'INPUT NUMBER OF STUD:'
      READ(*,*)N
      READ(*,*)(NUM(I),S(I),I=1,N)
      WRITE(*,110)
      DO 10 I=1,N
      WRITE(*,120)NUM(I),LEVEL(S(I))
10    CONTINUE
110   FORMAT(8X,'NO. STUD',4X,'LEVEL')
120   FORMAT(10X,I4,4X,A2)
      END
CC
CC
CC
      FUNCTION LEVEL(X)
      CHARACTER LEVEL,B(4)
      DIMENSION A(5)
      DATA A/0.0,60.0,80.0,90.0,101.0/
      DATA B/'D','C','B','A'/
      DO 10 I=1,4
      IF(X. GE. A(I). AND. X. LT. A(I+1))THEN
        LEVEL=B(I)
      END IF
      IF(X. EQ. 100)THEN
        LEVEL=B(4)
      END IF
10    CONTINUE
      END
```

若五个学生的学号和成绩为：

学号	101	102	103	104	105
成绩	56	60	100	89	77

则输出结果如下所示：

```
NO   STUD   LEVEL
     101      D
```

102　C

103　A

104　B

105　C

FORTRAN 77 规定允许函数子程序没有虚参，但必须保留括号，在引用函数子程序时，函数名后的一对括号也不能省略.

§10.2　子例行程序

子例行程序是一种比函数子程序应用范围更广的程序单位，它和函数子程序主要有两方面不同：

1. 函数子程序通常被设计成求一个函数值；而子例行程序被设计成可以得到更多的计算结果，也可以利用子例行程序来完成一系列与计算无关的操作.

2. 函数子程序的调用就像通常的函数引用一样，出现在表达式中. 而子例行程序则必须通过 CALL 语句来调用.

子例行程序的第一条语句是 SUBROUTINE 语句，其形式如下：

SUBROUTINE　子例行程序名($\mathbf{a}_1$,$\mathbf{a}_2$,…$\mathbf{a}_n$)

子例行程序名的取名规则和 FORTRAN 中其他名字的取名规则相同. 此名字仅仅是为了被调用，它不能像函数名那样可以作为变量在子程序的其他地方出现，子例行程序名不代表任何数值，因此与类型无关.

在子例行程序名后括号中的 a_1,a_2,…,a_n 是虚参，其形式和性质与函数子程序中的虚参完全相同. 但子例行程序与调用单位之间的数据传送，完全通过虚参和实参的结合来进行.

子例行程序可以没有虚参，这时 SUBROUTINE 语句就写成：

SUBROUTINE　子例行程序名

子例行程序通过 CALL 语句调用，调用形式如下：

CALL　子例行程序名($\mathbf{x}_1$,$\mathbf{x}_2$,…,$\mathbf{x}_n$)

此处 x_1,x_2,…,x_n 为与虚参对应的实参，它们应该与子例行程序中的虚参按位置在数量、类型和顺序上一一对应.

当程序执行到 CALL 语句时，进行虚实结合，程序转向去执行被调用的子程序中的语句，当遇到子程序中的 END 语句或 RETURN 语句时，就返回到 CALL 语句的下一条语句继续执行.

当子例行程序没有虚参时，调用形式为：

CALL　子例行程序名

子例行程序中的其他语句与函数子程序相同，因此不再重复.

现举例说明子例行程序的编写和调用.

假定输入 5 名学生的 3 门课程的成绩. 要求打印出以下成绩单：

```
-------------------------------------------------------------
-------------------------------------------------------------
NO. S      Q1      Q2      Q3      AVE
-------------------------------------------------------------
-------------------------------------------------------------
 11        60.0    70.0    80.0    70.0
-------------------------------------------------------------
 12        66.0    79.0    74.0    73.0
-------------------------------------------------------------
 13        70.0    82.0    91.0    81.0
-------------------------------------------------------------
 14        62.0    88.0    88.0    79.3
-------------------------------------------------------------
 15        80.0    79.0    86.0    81.7
-------------------------------------------------------------
-------------------------------------------------------------
```

其中第一栏是学号，最后一栏是每位学生的平均分，Q1，Q2，Q3 代表三门课程的成绩.

程序如下：

```
      PROGRAM MAIN
      DIMENSION Q(5,3),AVE(5),N(5)
      DO 1 I=1,5
         READ(*,*)N(I),(Q(I,J),J=1,3)
1     CONTINUE
      CALL MEAN(Q,AVE)
      CALL HEAD
      CALL TABLE(N,Q,AVE)
      CALL PRI
      END
CC
CC
      SUBROUTINE MEAN(S,A)
      DIMENSION S(5,3),A(5)
      DO 20 I=1,5
         SUM=0.0
         DO 10 J=1,3
            SUM=SUM+S(I,J)
10       CONTINUE
         A(I)=SUM/3.0
20    CONTINUE
      END
CC
CC
      SUBROUTINE HEAD
```

```
      CALL PRI
      CALL PRI
      WRITE( *,100)
100   FORMAT(12X,'NO. S',3X,'Q1',4X,'Q2',4X,'Q3',3X,'AVE')
      CALL PRI
      CALL PRI
      END
C
C
      SUBROUTINE TABLE(N,Q,AVE)
      DIMENSION Q(5,3),AVE(5),N(5)
      DO 10 I=1,5
        WRITE( *,100)N(I),(Q(I,J),J=1,3),AVE(I)
        CALL PRI
10    CONTINUE
100   FORMAT(11X,I5,4F6.1)
      END
C
C
      SUBROUTINE PRI
      WRITE( *,100)
100   FORMAT(10X,32('-'))
      END
```

在主程序中一共调用了四个子例行程序.

1. MEAN 子程序,用来求每位学生的平均分,调用时,把实参数组 Q 中存放的各门课程的成绩传送给子程序中的 S 数组.在子程序中,所得的各学生的平均分放在 A 数组中,由于 A 数组与主程序调用语句中的实参数组 AVE 对应,因此 A 数组中的值也一一对应地存放在 AVE 数组的各元素中.当 MEAN 子程序执行到 END 语句时,返回到主程序,接着执行 CALL HEAD 语句.

2. HEAD 子程序没有虚参,它的功能是打印表头.在这个子程序中,四次调用 PRI 子程序;每调用一次,打印一行短横线.

3. TABLE 子程序用来打印表格内容,它有三个虚参数组,N 中存放学号,Q 中存放学生成绩,AVE 中存放学生的平均分.当调用此子程序时,对应的实参把数据传送到子程序的各数组中,然后开始打印.每打印一行数据之后,就调用 PRI 子程序打印一行短横线.

4. PRI 子程序的功能就是打印一行短横线.

以上程序共有五个程序单位,程序结构如图 10.5 所示.在写程序时,子程序可以放在主程序后面,子程序书写的先后次序对程序的执行没有影响.

通过以上例题可以看到,在进行自顶向下、逐步求精的结构化程序设计中,子程序是一种很有用的工具,它能使一个庞大的程序按功能由一些子程序组成,这些子程序又可以再由一些子程

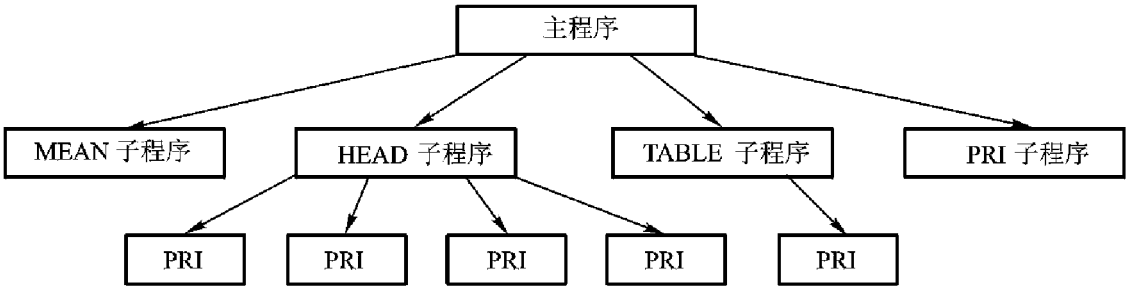


图 10.5

序组成，直到最底层的每个子程序用来完成一些最基本的操作为止. 按这种方法设计的程序，层次分明、易读易懂，也便于程序的维护和修改.

§10.3 虚 实 结 合

本节专门讨论各程序单位之间通过虚、实参数的结合所进行的数据传送. 现将可用作虚参和实参的各种名称列出如下：

虚　参	变量名(字符变量)	数　组	虚过程名	*
对应可用的实参	常数(字符串)， 变量名，数组元素，表达式	数组 数组元素	子程序名 内部函数名	可变返回点说明符

本书不讨论用 * 号作为虚参的有关内容. 用虚过程名作为虚参的有关内容将在本章第 4 节(211 页)中讨论.

10.3.1 用变量作为虚参

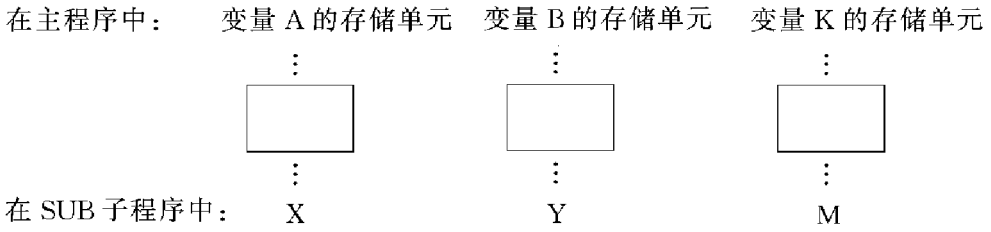
当虚参是一个数值变量时，它所对应的实参可以是类型相同的常数、变量数组元素和表达式. 现用以下主程序和子程序来举例说明虚实结合的过程：

```
PROGRAM MAIN                SUBROUTINE SUB(X,Y,M)
A=12.4                      A=0.5
K=5                         Y=X/2.0+A
CALL SUB(A,B,K)             M=M+1
WRITE(*,*)A,B,K             END
END
```

程序输出的结果如下：

```
12.4000000        6.7000000        6
```

当执行上述程序时，在主程序执行到 CALL 语句之前，SUB 子程序中的全部变量都没有确定的存储单元，因此称它们是无定义的. 当主程序执行到 CALL 语句时，控制转向子程序 SUB 去执行子程序中的语句. 同时，实参把它们的存储单元地址传送到子程序中，使对应的虚参和实参具有相同的地址. 这种结合可以用下图来形象地表示：



结合的结果实质上是使虚参 X 和实参 A 共用存储单元，虚参 Y 与实参 B 共用存储单元，M 和 K 共用存储单元. 由此可以看到，由于共用存储单元，因此，对应虚参和实参的类型必须一致，否则就会出错.

当控制转入子程序后，子程序中的变量 A 另辟一个临时的存储单元，与主程序中的变量 A 没有任何关系. 在子程序中，通过赋值语句，在 Y 的存储单元内存放了 6.7，在 M 的存储单元内存放了 6. 实际上就相当于使主程序变量 B 和 K 改变了原来的值，分别变成了 6.7 和 6. 当子程序的执行遇到 END 语句时，控制返回到主程序，去执行 CALL 语句的下一条 WRITE 语句，输出 A，B，K 的值. 从输出结果可以看到，B 和 K 的值已经改变. 主程序和子程序、调用单位和被调用单位之间就是通过这种虚实结合途径传送数据的（当然，函数子程序还通过函数名带回函数值）. 通常把以上这种虚实结合方式称为按地址结合.

在退出子程序后，子程序中所有的变量又都失去具体的存储单元而成为无定义的.

当虚参变量所对应的实参是数组元素时，因为数组元素也有固定的存储单元，因此虚实结合的情况与上相同.

当虚参变量所对应的实参是常数或表达式时，不同的 FORTRAN 编译程序会有不同的处理，为了避免出错，通常规定与常数和表达式对应的变量不得在子程序中重新赋值.

10.3.2 用数组作为虚参

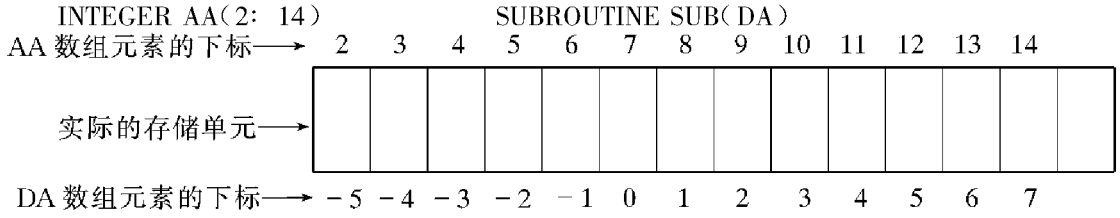
若虚参是一个数组名，则对应的实参必须是数组名或数组元素.

当两者都是数组名时，两个数组按地址结合. 实参数组的第一个元素和对应虚参数组中的第一个元素结合，第二个元素和第二个元素结合，其余依此类推. 例如，当有下面的调用和被调用语句时：

```
INTEGER AA(2: 14)                SUBROUTINE SUB(DA)
                                 INTEGER DA(-5: 7)
CALL SUB(AA)
```

实参数组和虚参数组的结合如下图所示：

上述子例行程序中对 DA 数组的定义，还可以改写如下：



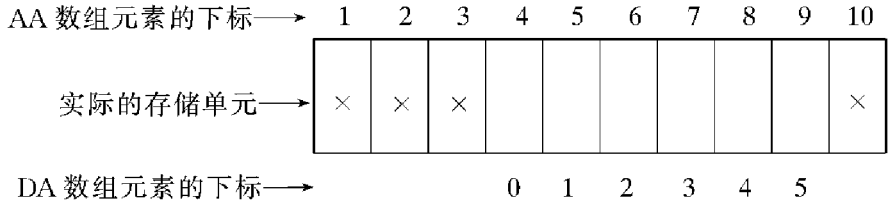
```
                                    INTEGER DA( -5: * )
        CALL SUB(AA)
```

AA 和 DA 数组的虚实结合与上图完全相同.

在子程序中,可以用 * 号作为虚数组的数组说明符中最后一个维定义符的上界.它的作用是使所定义的虚数组的大小和与之对应的实参数组大小完全相同,也就是说,当子程序未被调用时,虚数组的大小是假定的,假定它与所对应的实参数组大小相同.这种带有 * 号的数组说明符称为假定大小数组说明符,只能在子程序中对虚数组使用.

当与虚数组对应的实参是数组元素时,在虚实结合时,该数组元素把地址传送到子程序作为虚数组中第一个元素的地址,从而实现两个数组之间的虚实结合.于是,作为实参数组元素的下一个元素与虚数组中的第二个元素结合,其余依此类推.以下程序单位之间的虚实结合如下图所示:

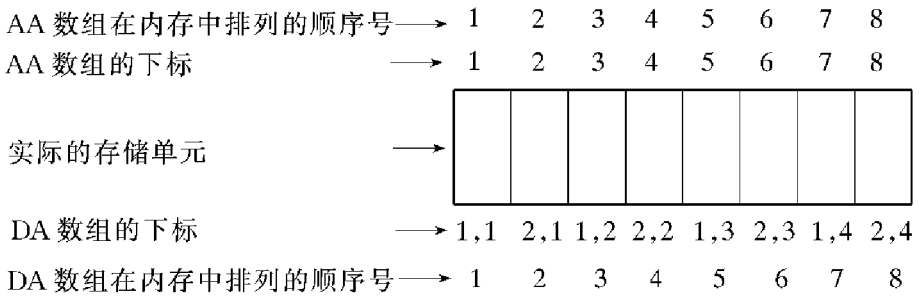
```
REAL AA(10)                 SUBROUTINE SUB(DA)
                            DIMENSION DA(0: 5)
CALL SUB(AA(4))
```



× 表示 AA 数组中未结合的部分

用作实参的数组,其说明符在维数、维的上下界、数组的大小方面,可以和与之结合的虚数组的数组说明符中所规定的不同.但是,实参数组的大小(即数组元素的个数)必须大于或等于与之结合的虚参数组的大小.同样,当与虚数组结合的实参是数组元素时,从此元素算起,该数组中元素的个数(包括此元素在内)必须大于或等于虚数组中元素的个数.因此,无论虚数组是与一个实数组结合还是与一个数组元素结合,虚数组的最后一个元素必须落在与之结合的实数组的范围之内.按照上述规定,以下的虚实结合是合法的.虚实结合的实际情况如下图所示:

```
DIMENSION AA(8)             SUBROUTINE SUB(DA)
                            DIMENSION DA(2,4)
CALL SUB(AA)
```



从上图可以看出,虚、实数组各元素的结合实际上是按数组元素在内存中排列的顺序一一对

应进行的. 当实参是数组名时，虚实结合按相同的顺序号进行对应数组元素的结合.

以下情况中，在实现虚实结合时都将发生错误，因为实参数组中没有足够的元素与虚数组中所有的元素结合.

```
      DIMENSION AA(2,4)          SUBROUTINE SUB(DA)
                                 DIMENSION DA(2,5)
      CALL SUB(AA)
```

或者

```
      DIMENSION AA(2,4)          SUBROUTINE SUB(DA)
                                 DIMENSION DA(2,4)
      CALL SUB(AA(2,1))
```

在程序设计中，应尽可能使对应的虚、实数组具有相同的维数，使每维具有相同的上下界，这样的程序清晰易读可避免出错，从以下的例子中可以看出这种设计的优点.

10.3.3 可调数组

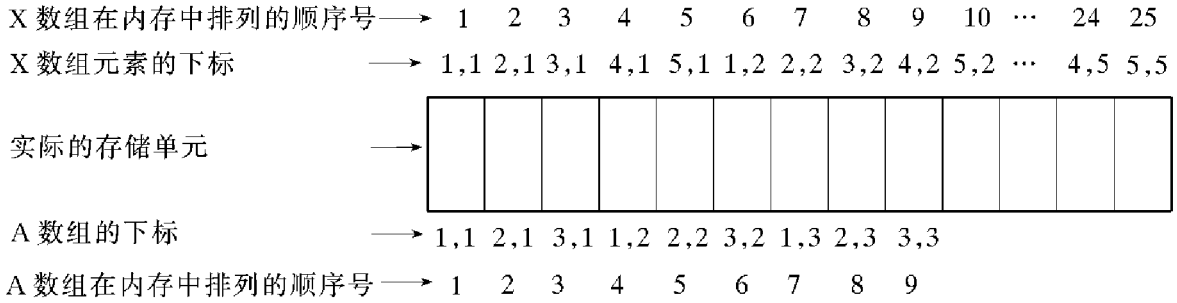
在对可调数组进行具体说明之前，我们先来分析下面的 IPR 函数子程序. 它的功能是求一个 3×3 数组中对角线上 1 到 N 个元素之积.

```
      FUNCTION IPR(A,N)
      INTEGER A(3,3)
      IF(N. LE. 0. OR. N. GT. 3)THEN
        WRITE(*,*)'ERR:',N
        IPR = -1
      ELSE
        IPR = 1
        DO 1 I = 1,N
          IPR = IPR * ABS(A(I,I))
    1   CONTINUE
      END IF
      END
C
C
C
C
      INTEGER X(5,5),A
      READ(*,*)X

      A = IPR(X,3)
      WRITE(*,*)A
```

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

从主观愿望上，希望通过主程序中 A = IPR(X,3)语句来求出 X(1,1) * X(2,2) * X(3,3)的值，当 X 数组中对角线上所有元素的值为 1，其余均为 0 时，X(1,1) * X(2,2) * X(3,3)的值应该为 1. 上面的程序从语法规则来说完全符合 FORTRAN 的各种规定，运行也能正常进行并最后输出结果. 但是输出的结果不是 1 而是 0. 现在我们用下图来分析虚实数组之间的结合情况，找出问题所在.



分析上图可以看出 A(1,1)与 X(1,1)结合，A(2,2)与 X(5,1)结合，A(3,3)与 X(4,2)结合. 因此在函数子程序中求 A(1,1) * A(2,2) * A(3,3)，对应于实参数组 X，实际上求得的是 X(1,1) * X(5,1) * X(4,2)的值. 由于只有 X(1,1)为 1，其余皆为 0，故乘积得 0. 需要注意的是，当 X 数组的值并不像所列举的那样有特点时，则所形成的错误是很隐蔽的，常常很难发现和查找.

从以上分析可以看出，IPR 函数子程序只对定义为 3 × 3 的实参数组才能真正求得对角线上 1 到 N 个元素的乘积，并不适用于任意大小的实参数组. 这种情况大大降低了对数组进行运算和操作的各种子程序的通用性.

为此，FORTRAN 提供了可调数组，允许在子程序中用变量来定义数组各维的下界和上界，以保证子程序中的数组能与对应的实数组在每一维中有相同的上下界，使虚数组的数组元素与对应的实数组的元素有相同的下标. 例如，可以把以上 IPR 子程序和调用它的主程序中的有关语句改写如下：

```
FUNCTION IPR(A,K,N)          INTEGER X(5,5),A
INTEGER A(K,K)
                             A = IPR(X,5,3)
```

当语句 A = IPR(X,5,3)调用 IPR 函数子程序时，由虚实结合 K 的值为 5，于是在子程序中定义了一个 5 × 5 的 A 数组. 这样一来，函数子程序 IPR 变成可以对任意大小的方阵求出对角线上 1 到 N 个元素之积.

可调数组在程序设计中是非常有用的工具，使用可调数组，大大提高了子程序的通用性，可调数组的使用，才使子程序真正成为结构化程序设计的重要手段.

使用可调数组必须遵循以下基本规则：

1. 可调数组的大小不得超过对应实参数组的大小.

2. 可调数组名必须作为虚参出现在子程序的第一条语句中.

3. 在可调数组说明符中,各个维说明符的下标上下界的表达式中,可以出现整型变量名,这种变量名必须出现在虚参表中或者出现在子程序的 COMMON 语句中(COMMON 语句的解释见第十一章).

10.3.4 虚参是字符型变量或字符型数组

当子程序中的虚参为字符型变量时,对应的实参必须是字符型变量、字符型数组元素、字符串常数、字符表达式. 字符型虚参必须在 CHARACTER 语句中进行说明. 长度说明方式与第八章中所述相同,这样说明的长度为固定长度. 若虚参的长度为固定长度,则它的长度必须小于等于对应实参的长度. 当长度小于实参长度时,从实参中最左边的字符开始与虚参结合,多余部分截去.

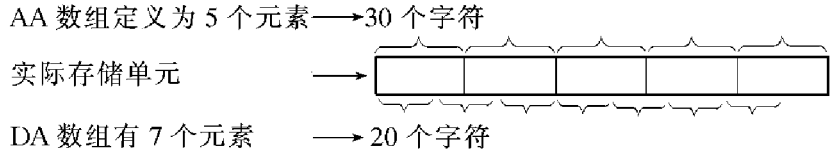
对字符型虚参变量还可以用 * 号来说明长度,这样说明的长度为假定长度. 当虚实结合时,虚参变量自动与对应的实参取相同的长度.

下面的函数子程序用来求出虚参变量 CH 中第一个字符到最后一个非空格字符的长度,函数值为一整型. 当 CH 全是空格时,函数值为 0.

```
      INTEGER FUNCTION LEND(CH)
      CHARACTER *(*) CH,SPACE*1
      PARAMETER(SPACE = ' ')
      DO 10 LEND = LEN(CH),1, -1
      IF(CH(LEND:LEND). NE. SPACE)RETURN
10    CONTINUE
      END
```

如果虚参是字符型数组,当虚数组元素的长度与对应实参数组中的元素长度一致时,它们的虚实结合情况与前面所述的数组虚实结合情况相同. 如果两个数组间元素的长度不同,则实参数组与虚参数组元素间并不一一对应相结合,其结合情况如下图所示.

```
      CHARACTER AA(5)*30              SUBROUTINE SUB(DA)
                                      CHARACTER DA(*)*20
      CALL SUB(AA)
```



因此,对虚数组大小的限制是:虚数组中字符的总数不得多于对应实参数组中的字符总数.

如果虚数组元素的长度用 * 号说明,则虚数组中每个元素的长度与对应实参数组元素的长度一致.

注意：当函数名或虚参的字符长度用 * 号说明时，这些函数名和虚参只能在赋值语句中参与并置运算，不能在任何其他语句中参与并置运算.

§10.4 EXTERNAL 语句和 INTRINSIC 语句（外部语句和内部语句）

10.4.1 过程

过程是指为得到问题的解答而执行的一步一步的动作或者子程序. 在 FORTRAN 程序中，以下内容统称为过程：

语句函数

内部函数

函数子程序（外部函数）

子例行程序

这些过程的名字称为过程名. 其中，函数子程序和子例行程序称为外部过程. 语句函数只能在本程序单位中引用，在虚实结合中不能作为参数进行传送.

10.4.2 过程名的虚实结合

首先，我们用程序举例来说明过程名的虚实结合. 该程序由五个程序单位组成，子程序 PR1 用来打印出 20 个 * 号，PR2 打印出 20 个 + 号，PR3 打印出 20 个 - 号. 在子程序 TRY 中，SUB 既作为子程序名出现在 CALL 语句中，又作为虚参出现在虚参表中. 因此 SUB 是一个虚过程名，它并不代表一个实际存在的外部过程. SUB 具体代表哪个外部过程要通过虚实结合才能确定. 显然，与 SUB 结合的实参应该是一个子程序名.

```
PROGRAM MAIN
EXETRANAL PR1,PR2,PR3

CALL TRY(PR1)
CALL TRY(PR2)
CALL TRY(PR3)
END
SUBROUTINT TRY(SUB)

CALL SUB

CALL SUB

END
SUBROUTINE PR1
WRITE( *,100)
```

```
100     FORMAT(1X,20('*'))
        END
        SUBROUTINE PR2
        WRITE(*,100)
100     FORMAT(1X,20('+'))
        END
        SUBROUTINE PR3
        WRITE(*,100)
100     FORMAT(1X,20('-'))
        END
```

在主程序中，在三条调用 TRY 子程序的语句中，分别用了三个子程序名作为实参. 当执行 CALL TRY(PR1)时，通过虚实结合，名字 PR1 与 SUB 结合，因此，在 TRY 中，CALL SUB 语句就相当于 CALL PR1 语句. 每次调用 TRY 子程序时，通过实参给 SUB 传送不同的子程序名，从而使得在 TRY 中调用了不同的打印子程序.

10.4.3 EXTERNAL 语句的使用

在 FORTRAN 程序中，主程序、子程序都是一些独立的程序单位，因此对上述程序中的主程序和各子程序都将单独进行编译. 在编译 TRY 时，由于名字 SUB 是作为外部过程名出现在 CALL 语句中，因此，编译程序将给虚参 SUB 赋以外部过程名的性质. 在编译主程序时，如果省略 EXTERNAL 语句，则编译程序将根据 PR1，PR2，PR3 在主程序中的位置定义它们为实型简单变量名. 这样，在执行程序时，由于在虚实结合的过程中虚、实参数的类型不一致而导致程序执行出错. 因此，当用外部过程名作为实参时，此名字必须在 EXTERNAL 语句中出现，以便定义它是一个外部过程名.

10.4.4 INTRINSIC 语句的使用

当把内部函数名作为实参时，必须用 INTRINSIC 语句对内部函数名进行定义.

EXTERNAL 语句和 INTRINSIC 语句都是说明语句，它们必须出现在可执行语句之前. 由于使用了这两条语句，因而，可以通过虚实结合传送外部过程名和内部过程名，这使程序设计者可以利用子程序这一手段，编写许多通用子程序，使 FORTRAN 子程序的用途变得更为广泛.

例 1 在 FORTRAN 中，所有三角函数的自变量都是以弧度为单位. 编写一个通用三角函数子程序 TRIG，只要在调用时给出函数名和角度(以度为单位)就能计算出三角函数的值.

TRIG 函数子程序如下所示：

```
        FUNCTION TRIG(X,F)
        R = X * 3.14159/180.0
        TRIG = F(R)
        END
```

如果需要计算 sin60°，cos30°，tan45°，只需用以下语句来调用即可：

```
INTRINSIC SIN,COS,TAN

X1 = TRIG(60,SIN)
X2 = TRIG(30,COS)
X3 = TRIG(45,TAN)
```

例 **2**　编写一个函数子程序，用来求任意函数的定积分.

利用所编的函数子程序求以下三个积分值.

$$y_1 = \frac{1}{\sqrt{2\pi}}\int_0^1 e^{-x^2/2}dx$$

$$y_2 = \frac{1}{2}\int_0^{\pi/2} \sin(x)dx$$

$$y_3 = \int_0^4 (x^2 + 3x + 2)dx$$

已知求任意函数定积分的梯形公式为：

$$\int_a^b f(x)dx = h\left[\frac{(f(a) + f(b))}{2} + \sum_{i=1}^{n-1} f(a + ih)\right]$$

此处 $h = \frac{(b-a)}{n}$，$i = 1,2,\cdots,n-1$.

求任意函数定积分的函数子程序 TRAP 如下所示，其中 FUN 是虚过程名. 要调用此函数求一个函数的定积分，必须通过虚实结合把该函数的名字传送给 FUN. 因此，TRAP 是一个通用的求积分函数子程序.

```
      FUNCTION TRAP(FUN,A,B,N)
      TRAP = (FUN(A) + FUN(B))/2.0
      H = (B - A)/REAL(N)
      DO 10 I = 1,N - 1
        TRAP = TRAP + FUN(A + I * H)
10    CONTINUE
      TRAP = TRAP * H
      END
```

为了求得函数 $e^{-x^2/2}$ 和 $x^2 + 3x + 2$ 的积分，必须把它们分别写成两个外部函数，在程序中用 PEXP(X)和 POLY(X)表示. 而在主程序中，外部函数名 PEXP 和 POLY 必须在 EXTERNAL 语句中定义. 内部函数名 SIN 则应该在 INTRINSIC 语句中定义. 当 TRAP 函数子程序已经存在时，以下是求 y_1、y_2、y_3 的完整程序.

```
      PROGRAM MAIN
      EXTERNAL PEXP,POLY
      INTRINSIC SIN
```

```
      Y1 = TRAP( PEXP,0.0,1.0,1000)/SQRT(2.0 * 3.1416)
      Y2 = TRAP( SIN,0.0,3.1416/2.0,1000)/2.0
      Y3 = TRAP( POLY,0.0,4.0,1000)
      WRITE( * , * )Y1,Y2,Y3
      END
CC
CC
      FUNCTION PEXP( X)
      PEXP = EXP( ( - X * * 2)/2.0)
      END
CC
CC
      FUNCTION POLY( X)
      POLY = X * * 2 + 3 * X + 2.0
      END
```

所求函数值分别为：

3.413445E - 001　　5.000014E - 001　　53.3333600

§10.5　SAVE 语句

前面已经讲过，当执行子程序遇到了 RETURN 语句或 END 语句后，子程序执行终止，子程序中所有变量、数组等都变成无定义. 也就是说，子程序在调用结束后，所有变量、数组的存储单元不再保留，在下次调用子程序时，这些变量和数组再重新赋值.

利用 SAVE 语句，可以把子程序执行过程中变量的值保存起来，等下次调用时，这些变量的值仍然存在并可使用. 例如，在本章第 1 节函数子程序的举例中（见 198 页），我们曾编写函数子程序求：

$$\mathrm{FUN}(x) = x + \frac{x^2}{1+2} + \frac{x^2}{1+2+3} + \cdots + \frac{x^n}{1+2+3+\cdots+n}$$

在 FUN 函数子程序中，每次都调用 SUM 函数子程序求每一项的分母；实际上，第 i 项的分母就等于前一项的分母 $1+2+3+\cdots+(i-1)$ 加上 i，因此，如果在 SUM 子程序中能保留上一次求得的和，这样的程序效率就高得多. 利用 SAVE 语句就可以达到这一目的，我们可以把 SUM 函数子程序改写如下：

```
      FUNCTION SUM( K)
      INTEGER SUM,S
      SAVE S
      DATA S/ 0/
      IF( S. EQ. 0)THEN
        DO 10 I = 1,K
         S = S + I
10      CONTINUE
```

```
      ELSE
        S = S + K
      END IF
      SUM = S
      END
```

SAVE 语句是一个说明语句，语句形式为：

```
      SAVE a1, a2, …, an
```

其中 $a_1, a_2, \cdots, a_n$ 是子程序中的变量名、数组名或/公用区名/. 如果 SAVE 语句后面什么也不写，则意味着包含本程序单位中所有允许在 SAVE 语句中出现的项目. 在 SAVE 语句中不允许出现函数名和虚拟参数名.

下面给出的是在程序中经常用到的随机函数 RAND. 在此函数子程序中，变量 SEED 用 SAVE 语句说明. 当函数子程序每次被调用时，SEED 得一个新的值，并且此值被保留，在下次调用 RAND 时，用 SEED 的值作为获得新的随机数的种子，而在函数子程序执行过程中，SEED 本身又获得一个新的值保留到下次被调用时使用. 程序如下：

```
      FUNCTION RAND(  )

      INTEGER SEED
      SAVE    SEED
      DATA    SEED/0/
      SEED = MOD(SEED * 29 + 217, 1024)
      RAND = REAL(SEED)/1024
      END
```

此随机函数得 0 到 1 之间的随机数(不包括 0 和 1).

§10.6 程序举例

例 1 编写一个函数子程序，求任意阶多项式 $a_0 + a_1x + a_2x^2 + \cdots +$ 的值.

函数子程序如下. 多项式的系数依次放在 A 数组中，a_0 放在 A(1)中，a_1 放在 A(2)中，…，N 代表系数的个数. 多项式的阶数为 N - 1.

```
      FUNCTION POLY(A, N, X)
      DIMENSION A(N)
      POLY = A(1)
      FX = 1
      DO 10 I = 2, N
        FX = FX * X
        POLY = POLY + A(I) * FX
10    CONTINUE
      END
```

可用以下主程序调用 POLY 函数，当 N 为 5，给 A 数组输入 7.0,6.0, -5.0,4.0,3.0，给 X 输入

2.0 时，Y 的值为 79.00.

```
      PROGRAM MAIN
      DIMENSION A(20)
      READ(*,*)N,(A(I),I=1,N)
      READ(*,*)X
      Y=POLY(A,N,X)
      WRITE(*,100)(A(I),I=1,N)
100   FORMAT(1X,10F6.2)
      WRITE(*,110)X,Y
110   FORMAT(6X,'X=',F6.2,'Y=',F6.2)
      END
```

例 **2**　一系列实验数据的标准误差可由以下公式确定：

$$\sigma=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$$

此处 $\bar{x}=\left(\sum_{i=1}^{n}x_i\right)/n$. 编写一个求标准误差的函数子程序 BETA.

BETA 函数子程序如下所示，其中 MEAN 也是一个函数子程序，用来求平均值 $\bar{x}$.

```
      FUNCTION BETA(A,N)
      REAL MEAN
      DIMENSION A(N)
      AMEAN=MEAN(A,N)
      SUM=0.0
      DO 1 I=1,N
        SUM=(A(I)-AMEAN)*(A(I)-AMEAN)
      CONTINUE
      BETA=SQRT(SUM/REAL(N-1))
      END
CC
CC
      FUNCTION MEAN(A,N)
      REAL MEAN
      DIMENSION A(N)
      SUM=0.0
      DO 1 I=1,N
        SUM=SUM+A(I)
1     CONTINUE
      MEAN=SUM/REAL(N)
      END
```

可用以下主程序调用 BETA 函数子程序进行试算.

```
      PROGRAM MAIN
      DIMENSION X(20)
      READ(*,*)N,(X(I),I=1,N)
      ST=BETA(X,N)
      WRITE(*,*)'THE STANDARD DEVIATION IS:',ST
      END
CC
```

例 **3**　编写函数子程序,求两个数的最大公约数.

现将求两数的最大公约数的算法叙述如下:设要求 a 和 b 的最大公约数. 首先将 b 除以 a,得商 q,得余数 r(如果 A < B,则将 A 和 B 对调),$a = bq + r$. 再将余数去除原来的除数,得新的商和余数,重复此过程,直到余数为零,则此时的除数就是 a 和 b 的最大公约数. 例如:求 1260 和 198 的最大公约数的步骤如下:

$$1\,260 = 198 * 6 + 72$$
$$198 = 72 * 2 + 54$$
$$72 = 54 * 1 + 18$$
$$54 = 18 * 3 + 0$$

(这时余数为 0,除数为 18)

因此,18 是 1260 和 198 的最大公约数.

下述 GCD 函数子程序用来求 X 和 Y 的最大公约数. 在函数子程序中,用 A 表示被除数,用 B 表示除数,R 表示余数.

```
      FUNCTION GCD(X,Y)
      INTEGER GCD,A,B,T,R,X,Y
      A=X
      B=Y
      IF(A. LT. B)THEN
        T=A
        A=B
        B=T
      END IF
      R=MOD(A,B)
10    IF(R. NE. 0)THEN
        A=B
        B=R
        R=MOD(A,B)
        GOTO 10
      END IF
      GCD=B
      END
```

可用下述主程序来调用 GCD 函数求出 N 个数的最大公约数.

```
      PROGRAM MAIN
      INTEGER A(20),GCD,B
      READ(*,*)N,(A(I),I=1,N)
      B=GCD(A(1),A(2))
      DO 1 I=3,N
        B=GCD(B,A(I))
1     CONTINUE
      WRITE(*,*)'THE SEQ OF NUMBER IS:',(A(I),I=1,N)
      WRITE(*,*)'THE GREATEST COMMON DIVISOR IS',B
      END
```

当求 1260,198,73 这三个数的最大公约数时,输出结果如下:

```
THE SEQ OF NUMBER IS:                    1260        198     72
THE GREATEST COMMON DIVISOR IS             18
```

例 **4**　编写一个子程序,用以对 N 个整数按照由小到大的顺序进行排序.

本例采用冒泡法排序,其算法如下:

从第一个数到第 N 个数两两进行比较,若前一个比后一个大,则进行交换. 重复此过程,直到不再需要进行交换为止. 现将此算法用框图表示在图 10.6 中

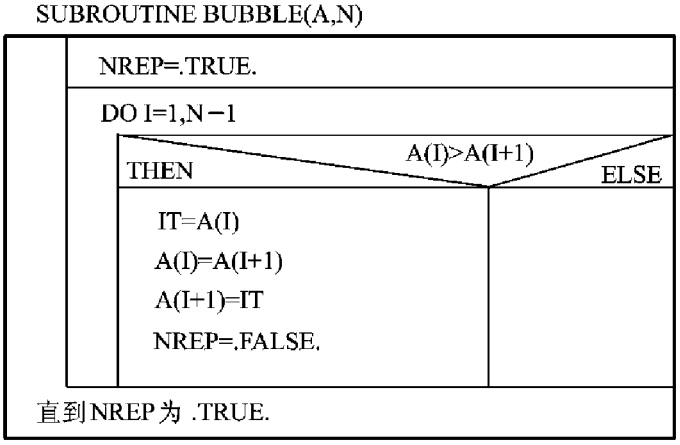


图 10.6

对以上算法再进一步分析可以发现,当进行第一趟两两比较时,必然将数列中最大的数放在第 N 个位置上. 因此,在进行第二趟比较时,只需在第一到第 N－1 个元素之间进行就可以了. 在进行完第二趟两两比较时,将把次大的数放在第 N－1 个位置上,因此,在进行第三趟比较时,实际上只需在第一到第 N－2 个元素之间进行,其他依次类推. 根据此分析,我们可以逐步缩小比较范围. 现将改进后的算法框图表示在图 10.7 中.

现将根据改进后的算法编写 BUBBLE 子程序,程序如下:

```
      SUBROUTINE BUBBLE(A,N)
      INTEGER A(N)
      LOGICAL NREP
      NUM=N
```

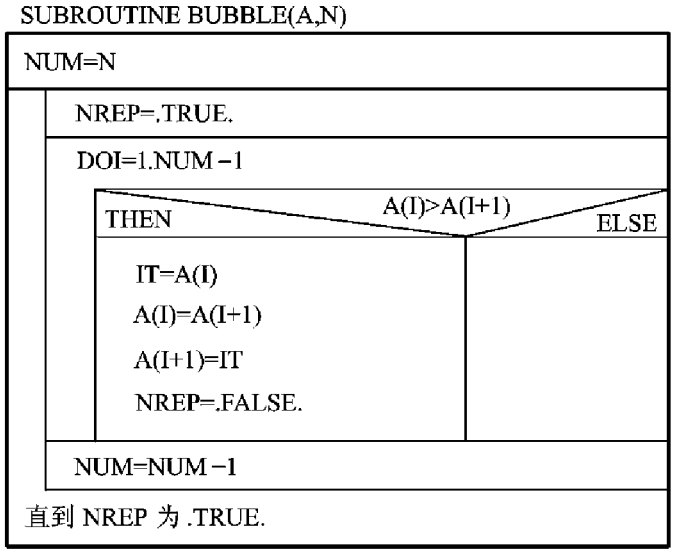


图 10.7

```
10      NREP = . TRUE.
        DO 20 I = 1, NUM - 1
          IF( A( I ). GT. A( I + 1 ))THEN
            IT = A( I )
            A( I ) = A( I + 1 )
            A( I + 1 ) = IT
            NREP = . FALSE.
          END IF
20      CONTINUE
        NUM = NUM - 1
      WRITE( * ,100)( A( K), K = 1, N)( 此语句在程序调试成功时应该删去)
100   FORMAT( 1X, 10I5 )
      IF( NREP )THEN
      ELSE
      GOTO 10
      END IF
      END
```

可用以下主程序调用 BUBBLE 子程序进行试算.

```
      PROGRAM MAIN
      INTEGER A( 50 )
      READ( * , * )N, ( A( I ), I = 1, N )
      WRITE( * , * )'THE SEQ OF NUM:', ( A( I ), I = 1, N )
      CALL BUBBLE( A, N )
      WRITE( * , * )'THE SCORE NUM:', ( A( I ), I = 1, N )
      END
```

当输入 8,3,5,2,6,9 这 6 个数时,输出结果是:

```
THE SEQ OF NUM:                    8     3     5     2     6     9
   3     5     2     6     8     9
   3     2     5     6     8     9   }这四行是子程序输出的中间结果
   2     3     5     6     8     9
   2     3     5     6     8     9
THE SCORE NUM:                     2     3     5     6     8     9
```

例 **5**　编写一个子程序，对输入的一串密码信息（不超过 80 个字符）进行密码翻译. 翻译的规则如下：凡是字母，都译成字母表中的下一个字母. 即 A 译成 B，B 译成 C，…，Z 译成 A. 其他字符不变.

子程序 TRANS 把密码信息串放在 A 数组中，逐个判断 A 数组每个元素的值. 若元素的值为字母 Z，则译成字母 A，若是其他字母，则调用 NEXT 函数子程序去译成字母表中的下一个字母.

函数子程序 NEXT 根据字符变量 C 中的字母来得到字母表中的下一个字母作为函数值.

子程序 TRANS 和 NEXT 的程序清单如下：

```
      SUBROUTINE TRANS(A,N)
      CHARACTER NEXT,LAST,C,A(N)
      DO 10 I=1,N
        C=A(I)
        IF(C. GE.'A'.AND. C. LT.'Z')THEN
         A(I)=NEXT(C)
        ELSE IF(C. EQ.'Z')THEN
         A(I)='A'
        END IF
10    CONTINUE
      END

      FUNCTION NEXT(C)
      CHARACTER NEXT,C,L(26)
      DATA L/'A','B','C','D','E','F','G','H','I','J',
     *       'K','L','M','N','O','P','Q','R','S','T',
     *       'U','V','W','X','Y','Z'/
      DO 10 I=1,25
        IF(C. EQ. L(I))THEN
         J=I+1
         NEXT=L(J)
        END IF
10    CONTINUE
      END
```

可用以下主程序调用 TRANS 子程序进行试算：

```
      PROGRAM MAIN
      CHARACTER A(80),B(80)
      WRITE(*,*)'INPUT N & STRING:'
      READ(*,*)N,(B(I),I=1,N)
      DO 1 I=1,N
      A(I)=B(I)
1     CONTINUE
      CALL TRANS(A,N)
      WRITE(*,*)'THE CODE:',(B(I),I=1,N)
      END
```

程序运行结果如下：

```
INPUT N & STRING:
14,'X','N','T','','Z','Q','D','','Q','H','F','G','S','!'(此行是输入值)
THE CODE:XNT ZQD QHFGS!
THE DECODE:YOU ARE RIGHT!
```

例 **6**　一行信息(最多 80 个字符)包含若干英文单词，单词之间用空格隔开. 选出一行中所有的单词放在数组中(假定每个单词最多由 12 个字母组成).

现用 FWORD 子程序来完成上述任务. 子程序 FWORD 把一串输入信息放在 LINE 中，把找到的单词放在 WORD 二维数组中，一行放一个单词. 变量 LP 指示 LINE 中当前所找的那个字符位置，NW 统计找到的单词数. FWORD 子程序的算法如图 10.8 中的框图所示：

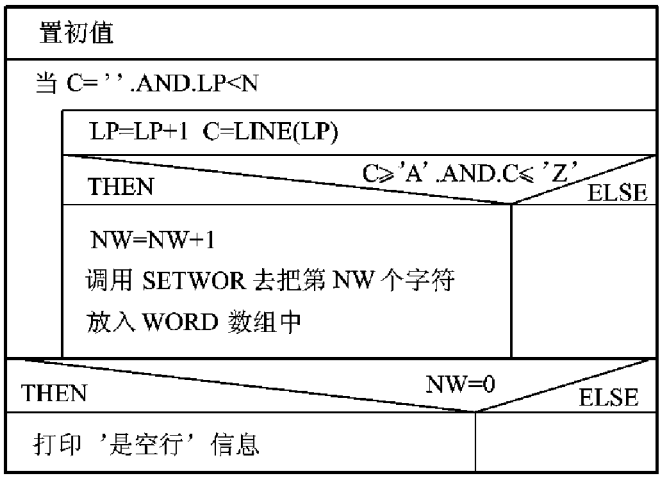


图 10.8

在其中调用了 SETWOR 子程序，把一个单词放在 WORD 数组中，SETWOR 子程序的算法如图 10.9 中的框图所示.

FWORDS 和 SETWOR　子程序清单如下：

```
      SUBROUTINE FWORDS(LINE,N,WORD,NN,L,NW)
      CHARACTER LINE(N),WORD(NN,L),C
```

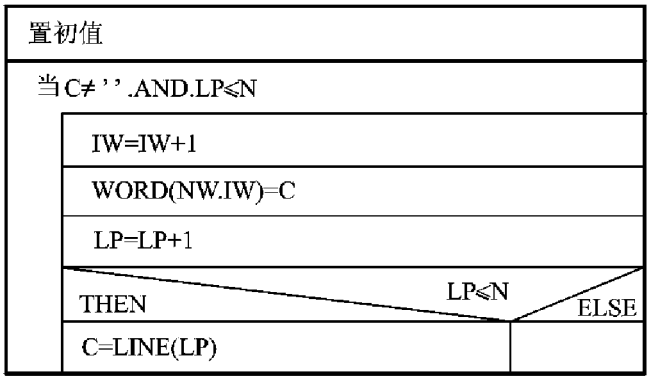


图 10.9

```
      DO 2 I = 1, NN
        DO 1 J = 1, L
         WORD( I, J) = ''
1       CONTINUE
2     CONTINUE
      LP = 0
      NW = 0
      C = ''
10    IF( C. EQ. ''. AND. LP. LT. N)THEN
        LP = LP + 1
        C = LINE( LP)
        IF( C. GE. 'A'. AND. C. LE. 'Z')THEN
          NW = NW + 1
          CALL SETWOR( LINE, N, WORD, NN, L, NW, C, LP)
        END IF
        GOTO 10
      END IF
      END
CC
CC
      SUBROUTINE SETWOR( LINE, N, WORD, NN, L, NW, C, LP)
      CHARACTER LINE( N), WORD( NN, L), C
      IW = 0
10    IF( C. NE. ''. AND. LP. LE. N)THEN
        IW = IW + 1
        WORD( NW, IW) = C
        LP = LP + 1
        IF( LP. LE. N)C = LINE( LP)
        GOTO 10
      END IF
      WRITE( *, *)'LP = ', LP, 'NW = ', NW, 'IW = ', IW
      WRITE( *, *)( WORD( NW, K), K = 1, IW)
      END
```

可用以下主程序调用 FWORDS 子程序. 已知一个单词最多由 12 个字母组成，一行中的单词若都是由一个字母组成，则最多为 40 个单词. 因此，定义 WORD 数组为 40 × 12 的二维数组，每行放一个单词，每个元素中放一个字母.

```
      PROGRAM MAIN
      CHARACTER LINE(80),WORD(40,12)
      READ( * ,100) N,(LINE(I),I=1,N)
      WRITE( * , * )'INPUT STRING IS:',(LINE(I),I=1,23)
      CALL FWORDS(LINE,N,WORD,40,12,NUMWOR)
      DO 10 I=1,NUMWOR
        WRITE( * , * )(WORD(I,J),J=1,12)
10    CONTINUE
100   FORMAT(I4/80A1)
      END
```

当从键盘上输入以下数据时：

```
27
THERE IS A BOOK ON THE DESK
```

运行结果为：

```
INPUT STRING IS:THERE IS A BOOK ON THE DESK
THERE
IS
A
BOOK
ON
THE
DESK
```

例 7　有 M 只猴子要选举猴王，选举方法如下：所有猴子排成一列，从头到尾报数，所报数能被 N 除尽者留下，其余退出. 留下者再从尾到头报数，所报数能被 N 除尽者留下，其余退出. 按上述规则反复报数，直到剩下少于 N 只猴子时，则此时报 1 者为王.

可用函数子程序进行上述选举，把选出为王的猴子号放在函数名 KING 中. 函数子程序的算法如下所示：

置 初 值	
	A 队列中 M 只猴子从 1 开始报数，一直报到 M；凡能被 N 除尽者站到另一队列 B 中，准备下一轮报数，记下 B 队中的猴子数 MM
	M = MM
	将 B 队列中的 M 只猴子，按逆序重新排列在 A 队列中。
直到　M < N	
KING = A(1)	

在 KING 子程序中调用 INVER 子程序来完成将 B 队列中的 M 只猴子按逆序重新排列在 A 队列中.

程序清单如下：

```
      PROGRAM MAIN
      INTEGER A(100),B(100)
      WRITE(*,*)'INPUT NUM OF MONKEY'
      READ(*,*)NUM
      WRITE(*,*)'INPUT NUM OF CYCLE'
      READ(*,*)N
      K=KING(A,B,NUM,N)
      WRITE(*,*)'THE KING IS:',K
      END
CC
CC
      FUNCTION KING(A,B,NUM,N)
      INTEGER A(NUM),B(NUM)
      DO 10 I=1,NUM
        A(I)=I
   10 CONTINUE
      M=NUM
   20   MM=0
        DO 30 I=1,M
         IF(MOD(I,N).EQ.0)THEN
          MM=MM+1
          B(MM)=A(I)
         END IF
   30   CONTINUE
        M=MM
        CALL INVER(B,A,M)
        WRITE(*,*)(A(I),I=1,M)
      IF(M.LT.N)THEN
      ELSE
      GOTO 20
      END IF
      KING=A(1)
      END
CC
CC
      SUBROUTINE INVER(B,A,M)
      INTEGER B(M),A(M)
      DO 10 I=1,M
        A(I)=B(M+1-I)
   10 CONTINUE
```

END

当程序运行时，若输入猴子数为 100，报数号为 7 的猴子留下，余者退出，则第 7 号猴子为王. 若输入猴子数为 31，报数号为 3 的猴子留下，余者退出，则第 24 号猴子为王.

习　　题

1. 已知

$$\text{norm} = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

编写子程序求 norm，n 由调用时给出.

2. 编写一求 n! 的子程序，n 由调用时给出.

3. 编写子程序求 $(x+y)^n$ 二项式系数：

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

当求 n 次的二项式系数时，k = 0 到 n. 子程序中调用求阶乘函数子程序来求阶乘.

编写程序求 n = 1 到 10 的各二项式的系数.

4. 编写子程序，求任意二次方程的根. 求二次方程根的公式如下：

$$x_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

5. 已知双曲正弦函数 $\sinh(x) = \frac{e^x - e^{-x}}{2}$，$e^x$ 可用以下级数近似表示：

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

（1）编写一个子程序，求 e^x，其中调用求阶乘函数.

（2）编写一个子程序，调用以上子程序求 sinh 函数.

（3）利用 SAVE 语句修改求阶乘函数子程序，使得不必每次被调用时都从 1 开始连乘，而只需对上一次求得的阶乘值再乘以自变量的值.

6. 编写子程序，求任意方阵的转置（即，原矩阵中的 a_{ij} 变成新矩阵中的 a_{ji}）.

7. 编写一个子程序，把任意直角坐标转换成极坐标. 编写一个子程序，把任意极坐标转换成直角坐标. 直角坐标和极坐标之间的转换公式如下：

$$x = \rho\sin\theta$$

$$y = \rho\cos\theta$$

8. 测量实验数据的失真系数可用以下公式求得：

$$\gamma = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^3}{n \cdot \sigma^3}$$

此处 $\bar{x}$ 是 $x_1, x_2, \cdots, x_n$ 的平均值. σ 是标准误差：

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

请编写子程序求 γ 值.

9. 某收音机厂生产四种型号的收音机，每种收音机中所用的电阻、电容和晶体管的数量如表所示：

收音机型号	电　　容	电　　阻	晶 体 管
1	2	6	3
2	6	11	5
3	13	29	10
4	8	14	7

电阻、电容、晶体管平均每个价格如下（以元为单位）：

电　　容	电　　阻	晶　体　管
1.0	0.2	1.4

编写一个通用子程序，能对包含有任意种元件的任意种型号收音机，求出成本费. 利用所编写的子程序，求以上4种型号每种收音机的元器件成本费.

10. 编写一个子程序，在一个字串中找出另一个字串出现的次数. 可以把字串放在一个数组中，每个元素内放一个字符.

11. 在名字表中找一个名字所在的位置，可以把名字放在一个二维字符数组中，一行放一个名字，每个元素放一个字符.

12. 编写一个子程序，找某个单词在字符串中的位置.

13. 编写子程序从N个猴子中选出猴王. 选择规则如下：先从1到N报数，凡是所报数能被M除尽者排除，余者留下再从尾到头报数，凡是所报数能被M除尽者排除，其余留下，再从头至尾报数. 重复以上过程，直到所剩猴子数少于M时，取最后一只为王.

14. 甲、乙、丙同时开始放鞭炮，甲每隔 t_1 秒放一次，乙每隔 t_2 秒放一次，丙每隔 t_3 秒放一次，每人各放N个鞭炮. 编写一个子程序，求出总共听到多少次鞭炮声.

15. 编写子程序，利用插入法进行排序. 假定待排序的数放在X数组中. 插入法排序的算法如下：先在A数组中放入第一项X(1)，然后把X(2)放在A(1)或A(2)的适当位置中；再把X(3)放在A(1)，A(2)或A(3)的适当位置中，使A(1)，A(2)，A(3)有序；其他依此类推. 当最后把X(N)放入A数组后，A(1)到A(N)中所有的数已按序排列.

16. 编写子程序，对任意数量的学生和任意门课程求平均分，并把学生学号和各成绩按平均分的优劣排好序.

第十一章　FORTRAN 中的其他语句

本章将简单介绍双精度型、复型数有关的说明语句，输入输出，算术 IF 和计算 GOTO 语句，共用存储单元的有关语句和数据块子程序. FORTRAN 中还包括另外一些使用较少的语句，如 ASSIGN 语句等，以及其他一些有关内容，本书将省略不作介绍，读者可在掌握了本书内容的基础上参考有关计算手册和其他书籍.

§11.1　双精度型运算和复型运算

11.1.1　双精度型运算

在 FORTRAN 程序中，实型数是一个近似值，一般提供七位有效位. 但在许多科学计算中，实型数的精度远不能满足要求，为此，FORTRAN 提供了双精度数. 一个双精度数在计算机内所占的存储量为实型数的一倍，通常它具有 15 到 17 位有效位. 双精度数也是一个近似值，只是精度更高，但运算速度较慢.

在同一个计算机系统中，双精度数与实型数的数值范围相同.

在 FORTRAN 程序中，双精度型常数用指数形式表示，只是必须把字母 E 换成 D. 以下是一些双精度数的表示形式：

数值	**FORTRAN** 双精度常数
0.0000001	1D－7
14.713	1.4713D1
0.0	0D0
1.0	1D0
3.141592653589798	3.141592653589798D0

请注意，在程序中写常数时，并不是位数越多就表示精度越高，因此所有的双精度常数都必须用以上右边的形式表示.

在 FORTRAN 程序中，存放双精度数的变量或数组都必须用以下语句说明：

DOUBLE PRECISION $v_1, v_2, \cdots$

其中 $v_1, v_2, \cdots$ 和其他说明语句中的内容相同.

在许多计算机的 FORTRAN 中可用

REAL＊8 $v_1, v_2, \cdots$

的形式来说明双精度型（在 IBM PC 机上只能用此语句说明双精度）.

在格式输入或输出中，双精度变量使用 D 编辑描述符，其形式为 nDw. d，其中 n 为重复系数，w 为字段宽度，d 为小数点后的位数.

11.1.2 复型运算

复型数在一些特殊的计算，如在电气工程方面的计算中有着广泛的用途. 例如，复数 6.5 + 18i 在 FORTRAN 程序中应表示成：

(6.5,18)

一对括号和两个数之间的逗号是 FORTRAN 复型常数的一部分，不能省略；括号中的第一个数表示复数的实部，第二个数表示复数的虚部. 在 FORTRAN77 中实部和虚部的数可以用实型常数或整型常数表示（而在 FORTRANⅣ中只能用实型常数）. 注意，在复型常数的一对括号中绝不允许出现变量名或算术表达式；也不能出现符号常数. 因此下述标号为 10 的赋值语句的右边是错误的.

```
      COMPLEX A
      X = 6.5
      Y = 18
10    A = (X,Y)
```

以下是数学上复型数与 FORTRAN 复型数的对照：

数值	FO RTRAN 复型数
2i	(0.0,2.0)
1	(1.0,0.0)
100 - 100i	(100.0, -100.0)

在 FORTRAN 程序中，存放复型数的变量或数组都必须用以下语句说明：

COMPLEX $v_1, v_2, \cdots$

在 FORTRAN 中，复数可进行各种算术运算. 如果 A1 = (a_1, b_1)，$A_2 = (a_2, b_2)$，此处 a_1, b_1, a_2, b_2 代表实常数. 则

$A1 + A2 = (a_1 + a_2, b_1 + b_2)$

$A1 - A2 = (a_1 - a_2, b_1 - b_2)$

$A1 * A2 = (a_1 * a_2 - b_1 * b_2, a_1 * b_2 + a_2 * b_1)$

$A1/A2 = ((a_1 * a_2 + b_1 * b_2)/(a_2 * a_2 + b_2 * b_2), (a_2 * b_1 - a_1 * b_2)/(a_2 * a_2 + b_2 * b_2))$

每个复型数的输入和输出用两个实型编辑描述符，下述程序列举了复型数的输入和输出.

```
      PROGRAM MAIN
      COMPLEX A,B,C
CC READ   TWO COMPLEX NUMBERS
      READ( *,'(2F10.3,2F10.3)')A,B
      C = A * B
CC PRINT  DATA ITEMS AND THEIR PRODUCT
      WRITE( *,100)A,B,C
  100 FORMAT(1X,'A = ','(',F10.3,',',F10.3,')'/
     *       1X,'B = ','(',F10.3,',',F10.3,')'/
     *       1X,'C = ','(',F10.3,',',F10.3,')')
```

```
      END
```

与复型数有关的函数有 CMPLX(e_1,e_2),e_1 和 e_2 可以是任意类型的算术表达式,此函数用来把 e_1 和 e_2 的值分别转换成复数的实部和虚部. AIMAG(Z)用来取复数 Z 的虚部,REAL(Z)用来取复数 Z 的实部,CONJG(Z)用来求复数 Z 的共轭复数.

§11.2 算术 IF 语句和计算 GOTO 语句

11.2.1 算术 IF 语句

算术 IF 语句的形式如下:

$$\mathbf{IF(e)}s_1,s_2,s_3$$

其中 e 是一个算术表达式,s_1,s_2,s_3 分别代表本程序单位中的三个语句标号. 当 e 的值小于零时,控制转向标号为 s_1 的语句去执行,当 e 的值等于零时,控制转向标号为 s_2 的语句去执行,当 e 的值大于零时,控制转向标号为 s_3 的语句去执行. 例如,按以下公式计算 y 的值:

$$y=\begin{cases}\dfrac{1}{x}(x>1)\\0(x=1)\\x(x<1)\end{cases}$$

可写出程序如下:

```
      READ(*,*)X
      IF(X-1.0)10,20,30
10    Y=1.0/X
      GOTO 40
20    Y=0.0
      GOTO 40
30    Y=X
40    WRITE(*,*)'X=',X,'Y=',Y
      END
```

11.2.2 计算 GOTO 语句

计算 GOTO 语句的形式如下:

$$\mathbf{GOTO}(s_1,s_2,\cdots,s_n),e$$

其中 e 是一个整型表达式,e 的值应该为 1 到 n. 当 e 的值为 1 时,转向第一个标号 s_1;当 e 的值为 2 时,转向第二个标号 s_2,…,当 e 的值为 n 时,转向第 n 个标号 s_n. 当 e 的值小于 1 或大于 n 时,接着执行此 GOTO 语句的下一条语句. 例如:

```
READ(*,*)I
IF(I. GE. 5. OR. I. LT. 1)THEN
```

```
      WRITE( * , * )'ERR VALUE:',I
      STOP
    END IF
    GOTO(100,200,210,220,300),I
```

§11.3 EQUIVALENCE 语句和 COMMON 语句

11.3.1 EQUIVALENCE 语句(等价语句)

EQUIVALENCE 语句是一个说明语句.它的主要用途是节约内存空间,使一些不交叉使用的变量和数组共用存储单元;它的另一用途是为程序设计提供一种方便手段,即可以用一个名字来顶替另一个名字.

EQUIVALENCE 语句的一般形式是:

EQUIVALENCE(名字表),(名字表),…

每个名字表用一对括号括起来,各名字表之间用逗号隔开.名字表中可以包含变量名、数组元素名、数组名和字符子串名.一个名字表中至少要包括两个名字,但不能包括函数名和作为虚参的名字.

在 EQUIVALENCE 语句中的所有名字都必须出现在同一个程序单位中.语句的功能就是使每个名字表中的名字共享存储单元.例如:

```
EQUIVALENCE(A,B),(D,E,F)
```

上述语句使变量 A 和 B 共用一个存储单元,使变量 D,E 和 F 共用一个存储单元.若有以下语句

```
INTEGER X(8),Y(8),L(12),A(3),B(7),C(2)
EQUIVALENCE(L,A,X),(X(6),B(3)),(Y(5),L(11),C)
```

在名字表(L,A,X)中,所有名字都是数组名,它意味着三个数组从第一个元素起共用存储单元.因此,此名字表相当于(L(1),A(1),X(1)).因为每个数组中的所有元素都占一系列连续的存储单元,所以当各数组的第一个元素共用一个存储单元时,各数组的第二个元素也必然共用存储单元,其他依此类推.名字表(X(6),B(3))表示 X 数组的第六个元素与数组 B 的第三个元素共用存储单元.因此,按上述数组在内存中排列规则,X(7)和 B(4)将共用一个存储单元,X(8)和 B(5)共用一个存储单元.上面的 EQUIVALENCE 语句最终形成的内存分配情况如下图所示.图中每一个框代表一个存储单元.

L(1)	L(2)	L(3)	L(4)	L(5)	L(6)	L(7)	L(8)	L(9)	L(10)	L(11)	L(12)		
A(1)	A(2)	A(3)	B(1)	B(2)	B(3)	B(4)	B(5)	B(6)	B(7)				
X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)			C(1)	C(2)		
						Y(1)	Y(2)	Y(3)	Y(4)	Y(5)	Y(6)	Y(7)	Y(8)

从图中可以看到，由于以上语句，使 L(7)、B(4)、X(7)、Y(1)共用一个存储单元，而且这四个数组元素类型相同，因此当 Y(1)在程序中得到一个值，例如 Y(1)=22 时，则 X(7)，B(4)，L(7)也都具有相同的值 22.

EQUIVALENCE 语句的一个名字表中的名字，可以有不同的类型. 这时，当一个名字有值时，另一个不同类型的名字并无定义. FORTRAN77 规定，字符类型的名字不能与其他类型的名字出现在同一个名字表中，也就是说，字符类型只能与字符类型等价. 例如：

```
CHARACTER A*4,B*3,C(2)*2
EQUIVALENCE(A,B,C)
```

其共用存储单元的情况如下所示：

A(1: 1) B(1: 1) C(1)(1: 1)	A(2: 2) B(2: 2) C(1)(2: 2)	A(3: 3) B(3: 3) C(2)(1: 1)	A(4: 4) C(2)(2: 2)

在使用 EQUIVALENCE 语句时，要避免出现矛盾的等价关系. 例如：

```
DIMENSION A(10)
EQUIVALENCE(A(2),C),(A(3),C)
```

上述语句规定 A(2)和 A(3)共用存储单元，但 A(2)和 A(3)是同一数组中的元素，必然占连续两个不同的存储单元，因此出现了矛盾，所以上述的等价关系是错误的.

EQUIVALENCE 语句常用于使一些非常大的数组共享存储单元，以便节省内存空间，而并非为了使它们的值等价. 例如：

```
DIMENSION A(10000,2),N(10000)
EQUIVALENCE(A,N)
```

当程序中某一段需要使用 A 数组而并不同时使用 N 数组，并且在过后 A 数组又不再使用时，可用以上语句让 N 数组占用 A 数组原来的存储空间，从而节省了 10000 个存储单元. 注意，不要同时使用 A 和 N 数组(除非经验非常丰富的程序员)，否则将会引起混乱.

另外，由于某种原因而需要用一个名字来代替另一个名字时，可利用 EQUIVALENCE 语句. 如：

```
EQUIVALENCE(NR,MR)
```

这时，同一名字表中的名字类型必须相同，因而起到真正的“等价”作用.

11.3.2 COMMON 语句(公用语句)

在子程序一章中，我们曾经叙述过在各程序单位之间通过虚实结合的方式进行数据传送. 此处所介绍的 COMMON 语句是用在各程序单位之间进行数据传送的另一途径. COMMON 语句采用使不同程序单位的变量共享存储单元的方法，来达到在不同程序单位之间进行数据传送的目的. COMMON 语句是一条说明语句，语句形式是：

```
COMMON/公用区名/名字表,…,/公用区名/名字表
```

公用区名是为某个数据区指定的名字，它以字母开头，最多由六个字母、数字组成. 名字表中只能出现变量名、数组名和数组说明符. 下述左右两边语句的作用是相同的.

```
DIMENSION A(10)
COMMON/NAME/A,B          COMMON/NAME/A(10),B
```

在一个程序单位中，在同一个有名公用区中，所有变量、数组元素均按它们在 COMMON 语句中出现的顺序，先后放在此公用区的存储区域中，也就是说，在一般情况下，我们不可能知道变量存放在计算机内存中的什么地方. 而使用 COMMON 语句，例如在下面的主程序段中，我们就可以指定变量 A 放在名字为 T1 的存储区的第一个存储单元中，B 放在该存储区的第二个单元中，X(1)到 X(10)放在该存储区的第三到第十二个存储单元中，N1，N2 放在该存储区的第十三、第十四个存储单元中. 在子程序中，我们命 A1 放在 T1 存储区的第一个存储单元中，因此 A1 和主程序中的 A 必然共用一个存储单元，其他依此类推.

```
PROGRAM MAIN
COMMON/T1/A,X(10),N1,N2,/T2/I,J,C(4)

CALL SUB(ARY,N,M)

SUBROUTINE SUB(XARY,N1,M1)
DIMENSION XARY(N1,M1)
COMMON/T1/A1,X1(5),/T2/I1,J1,C(4)
COMMON/T1/Y(5),NA,NB
```

上面两个程序单位在 T1 和 T2 公用区中存储分配情况如下：

T1 公用区

A	X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)	X(9)	X(10)	N1	N2
A1	X1(1)	X1(2)	X1(3)	X1(4)	X1(5)	Y1(1)	Y1(2)	Y1(3)	Y1(4)	Y1(5)	NA	NB

T2 公用区

I	J	C(1)	C(2)	C(3)	C(4)
I1	J1	C(1)	C(2)	C(3)	C(4)

从上图可以看出，使用 COMMON 语句可以通过利用共同的存储区使两个不同程序单位中的变量进行结合. 因此，当两个程序单位之间需要有大量数据进行传递时，使用 COMMON 语句就不必在每次调用时一一写出传送的数据项，而只需在本程序单位中写一条 COMMON 语句就能实现. 不同程序单位中同一公用区中的变量按在 COMMON 语句中出现的先后次序一一对应结合，这种结合称为公用区结合. 不同类型的变量虽然可以占用共同的存储单元，但是实际上起不到传递数据的作用，因此建议在使用时使同一公用区中变量类型一一对应，以免引起不必要的麻烦.

具有名字的公用区称为有名公用区，不具名字的公用区称无名公用区. 使用有名公用区时应该遵循下述规则：

1. 公用区名不能与本程序单位中的主程序名、符号常数名、内部函数名、外部过程名和数据块子程序名同名，但可以和变量、数组同名.

2. 不同程序单位中，同名公用区的长度必须一致.

3. 程序单位中，一个名字只能在一个公用区中出现一次.

4. 在一个公用区中若出现字符型变量时，则同名公用区中所有名字（数组名、变量名）都必须是字符型的，也就是说，字符型变量不允许与数值型变量在同一公用区中出现.

5. 虚参不允许出现在公用区中.

6. 公用区中的变量不允许用 DATA 语句赋初值，有名公用区中的变量要赋初值时，必须用 BLOCK DATA 子程序实现，这将在下一节中介绍.

当在两条斜杠之间省略公用区名时，这时形成无名公用区. 例如：

```
COMMON//M,Q(3,3),//A,B
```

此语句也可以写成：

```
COMMON M,Q(3,3),A,B
```

一个程序中只有一个无名公用区. 在不同的 COMMON 语句中，出现在无名公用区或同名公用区中的变量、数组，按出现的先后顺序排列在无名公用区或同名公用区中. 因此下面的语句所定义的在无名公用区中的内容，实际上和上面语句所定义的相同.

```
COMMON M,/T1/I,J,K
COMMON /T2/P(10),//Q(3,3)
COMMON A,B
```

无名公用区和有名公用区不同之处在于：

1. 各程序单位中的无名公用区中的长度可以不一致.

2. 无名公用区中的成员不能用任何方式赋初值.

11.3.3 COMMON 语句和 EQUIVALENCE 语句联用

COMMON 语句和 EQUIVALENCE 语句联合使用可以扩充公用区. 例如：

```
PROGRAM MAIN                    SUBROUTINE SUB
DIMENSION MY(6)                 COMMON M(3),NP(4)
COMMON I,J,K,IT(4)
EQUIVALENCE(K,MY(1))

CALL SUB
```

这时无名公用区中存储分配的情况如下图所示：

I	J	K	IT(1)	IT(2)	IT(3)	IT(4)	
		MY(1)	MY(2)	MY(3)	MY(4)	MY(5)	MY(6)
M(1)	M(2)	M(3)	NP(1)	NP(2)	NP(3)	NP(4)	

由于 K 和 MY(1)等价，致使 MY 数组的元素也被带入公用区，使得 IT(1)与 MY(2)共用一个存储单元，其他元素一一对应，并使公用区向后延伸一个存储单元.

子程序中规定 M(1)放在无名公用区中的第一个存储单元，因此与主程序中的 I 共用一个存储单元. 其他如上图所示.

如果程序中有以下语句：

```
DIMENSION MY(6)
COMMON I,J,K,IT(4)
EQUIVALENCE(K,MY(4))
```

这时存储分配将出现以下情况：

	I	J	K	IT(1)	IT(2)	IT(3)	IT(4)
MY(1)	MY(2)	MY(3)	MY(4)	MY(5)	MY(6)		

↑
无名公用区的起点

由上图可见，MY(1)向前越过了无名公用区的起始位置，这样扩充公用区是错误的. COMMON 语句和 EQUIVALENCE 语句联合使用，允许向后扩充无名公用区，而不允许向前扩充公用区. 因此要避免出现上面这种错误.

同名公用区中的变量和不同名的有名公用区中的变量，不能出现在 EQUIVALENCE 语句的同一个名字表中，或通过 EQUIVALENCE 语句间接地导致共用存储单元.

§11.4 BLOCK DATA 子程序(数据块子程序)

BLOCK DATA 子程序专门用来给有名公用区中的变量赋初值. 此程序以下面的语句开头：

BLOCK DATA 名字

用 END 语句结束.

在 BLOCK DATA 子程序中没有可执行语句，全部都是用来说明有名公用区中成员的说明语句，另外还有 DATA 语句. 例如：

```
BLOCK DATA AA
DIMENSION A(4),B(4)
INTEGER NUM(10),ARRAY(5,5)
COMMON/TABLE/A,B,C,I,J,K,NUM,ARRAY
DATA NUM/1,2,3,4,5,6,7,8,9,10/
END
```

即使只给有名公用区 TABLE 中的某一个成员赋初值，也必须在 COMMON 语句中把 TABLE 公用区中的全部成员列出.

在同一个有名公用区中的成员若要赋初值时，只能在一个数据块子程序中进行，而不能由不同数据块子程序对一个有名公用区中的成员赋初值.

在同一个程序中可以有多个数据块子程序.数据块子程序名不能与其他主程序名、外部过程名、公用区名和其他数据块子程序同名.数据块子程序也可以没有子程序名，但这种无名的数据块子程序在一个程序中只能有一个.

第十二章　文　　件

通过前面各章的学习，读者已经掌握了用 FORTRAN77 语言进行程序设计的基本知识，并能根据题目的要求设计一些简单的程序. 我们可以发现，在我们所介绍的或读者所运行的程序中大致都包括几个主要步骤：输入数据，对数据进行处理，输出数据，最后结束程序的运行. 当然，输出的数据是在终端上显示的（或用打印机打印出来），而输入的数据则由终端键盘输入. 因此，一旦程序结束，除了得到一份打印结果外，计算机未保留任何结果. 我们尚未涉及到这样的情况：即，把本次运行的结果保存在计算机的某种外部介质（例如磁带或磁盘）上，在下一次（或下一个）程序运行时，直接从这些介质上读入数据，而不必再通过终端敲入数据. 在很多情况下，这是非常有用的. 现在来分析一个最简单的例子：如果我们要对一千名学生某门课程的成绩进行处理，这时至少需要输入一千个数据. 当从键盘敲入数据时，如果敲错一个数据，程序运行结果就不正确，必须重新再送一千个数据，直到所有数据都准确无误地输入为止. 如果数据更多，敲错数据的可能性就越大，每次出错，都必须重新开始送入，这样做的工作量必然很大. 如果我们先编写一个程序来读入一千个学生的数据，把它们存放在外部介质上，同时通过打印出输入数据，来检查输入是否正确，如果发现有错，再通过程序对介质上个别错误进行修改，直到确认所有数据正确无误为止. 最后才从外部介质上读入所有数据来进行处理，这样做的效率显然要高得多.

在程序中可以使用 READ 语句把外部介质上的数据读入到计算机内存中，用 WRITE 语句把数据传送到外部介质上，其形式与通过标准输入输出设备（如键盘和终端显示器）传送数据的形式相似. 但是，在对此进行详细介绍之前，我们必须建立两个重要的概念——记录和文件.

FORTRAN 程序输入和输出的记录由一系列字符或数据组成. 通过 WRITE 和 READ 语句输出和输入的记录有两种，它们是有格式记录和无格式记录.

文件由一系列记录组成，在 FORTRAN 中有两种数据文件：内部文件和外部文件. 本书将只讨论外部文件.

一个外部文件是存储在某些外部介质上的、可识别的一系列记录. 因此一份打印清单可以看成是一个文件，一叠穿孔卡片也可以看成是文件；在计算机系统中也把某些外部设备指定为文件，如打印机、卡片机. 但平常所说的文件，总是指保存在磁盘或磁带上的那些文件.

文件分顺序存取文件和直接存取文件两种. 顺序存取文件的特点是：由程序输出记录时，总是从文件的开始处起，一个接着一个，按输出的顺序写（存放）在外部介质上. 当需要读入某个记录时，也必须按写时的记录顺序逐个读入核记录前的所有记录. 顺序文件可存放在磁带或磁盘上，但磁带上只能存放顺序文件.

直接存取文件又称随机存取文件，用户可根据需要直接存取某个指定的记录，而不必每次都从文件的开始处进行存取. 因此，这种文件的存取速度快，具有较大的灵活性. 目前，磁盘是存放

直接存取文件的较理想的介质.

当前,最常用的计算机外部存储介质是磁带和磁盘,这些外部介质的特点是体积小,存储的信息量大.典型的磁带,一盘长度可达2400英尺以上,可包含五千万个字符或更多数量的信息.磁盘包含的信息量更大,一片5英寸的软磁盘可包含几十万甚至上百万个字符信息量,一个固定磁盘组则可存放500兆以上字符或同等信息量,而且存取信息的速度迅速、方式灵活.

为了便于初学者学习起见,本章将只介绍与磁盘文件有关的语句和内容,并不打算全面介绍FORTRAN所包含的与文件有关的全部语句.读者在掌握了本章内容之后,可以根据需要,参考有关书籍或说明书,进一步了解其他内容.

§12.1 有格式记录和无格式记录

记录,是FORTRAN输入输出的基本单位.在FORTRAN中,有格式记录必须用格式输出语句写,用格式输入语句读.无格式记录只能用无格式输出语句写,无格式输入语句读.记录具有长度,记录的长度以字节为单位,格式记录中,一个字符占一个字节.例如,格式输出时,记录中含有80个字符,则记录长为80字节.

在格式输入输出中,开始一个新的记录的方式可概括如下:

1. 每执行一次READ语句或WRITE、PRINT语句就开始输入或输出一个新的记录.

2. 在格式输入输出中,遇到以下两种情况就会开始一个新的记录:

其一,每当在格式说明中遇到一个斜扛号(/)则开始一个新的记录.

其二,当输入输出语句数据表中的项目还未输入输出完毕,并且对应的格式说明已达到最右边的括号,而需要重新开始使用格式说明时,则开始一个新的记录.

例如,以下语句把一个记录送到设备号为2的文件上,此记录的位置紧接在上一次写的记录后面.

```
      WRITE(2,100)A,B,C,D
100   FORMAT(4F10.3)
```

记录的长度为40个字节,因为每个输出数据占10个字符宽.在此,我们暂且用设备号来代表某个文件,有关内容将在下一节中介绍.

以下语句把4个记录连续送到设备号为3的文件中,其中第一个记录放在第100个记录中,记录长度为4.

```
      WRITE(3,100,REC=100)I,J,K,L
100   FORMAT(I4)
```

以下语句从设备号为8的文件中读入一个记录.

```
      READ(8,'(2I7,2F10.3)')I,J,A,B
```

该记录的长度必须大于44,否则会发生错误.

以下表控输出语句将一个记录送到设备号为1的文件中去.

```
      WRITE(1,*)A,I,B,J
```

由于表控输出语句的格式由各计算机系统决定,并且往往要到输出时才能确定记录长度.因此,

表控数据文件没有通用性，而且对于那些需要预先确定记录长度的文件来说，使用表控输出语句输出数据到文件中是不合适的.

程序中各变量的值在计算机内部是用二进制的机器代码表示的. 计算机在执行格式输出时，实际上要完成两项功能，首先把输出表中各项的值按照格式转换成一串字符，然后把这些字符输出到指定的文件(或设备)中. 在输入时，计算机再按指定的格式从文件中读入一系列字符，然后转换成计算机内部的数据. 在许多情况下，把数据写到文件上去的目的并不是为了让人们读它，而主要是作为以后所执行程序的输入数据，让计算机读它. 在这种情况下，当输出时把内部数据转换成一串字符，输入时再把一串字符转换成内部数据的步骤是多余的. 为了避免这些多余的步骤，提高输入输出执行的效率，可以使用无格式读、写语句来输入输出无格式记录. 无格式输入输出语句与格式输入输出语句的主要区别是语句中不存在格式标识符. 最简单的无格式输入输出语句的形式是只指定设备号，如：

```
WRITE(9)X,Y
```

通过无格式 WRITE 语句输出到文件中的不是一系列字符，而是代表数据的一系列二进制代码序列，即按计算机内存中数值的存储形式输出. 由无格式 WRITE 语句输出到文件中的值，只能用无格式 READ 语句读入.

每执行一次 WRITE 语句，在文件上就建立一个记录，无格式数据文件中记录的建立只与 WRITE 语句的执行有关. 每执行一次 READ 语句，就将在指定的文件上读入一个记录.

无格式文件中的记录长度与传送数据的类型有关. 通常，一个实型或一个整型数占 4 个字节，一个双精度型数或一个复型数占 8 个字节，字符型变量中的一个字符占一个字节，一个逻辑值占一个字节.

在任何文件中，所有记录要么都是无格式的，要么都是有格式的. 而无格式输出所建立的文件只能存储在磁带或磁盘上.

如果在一种计算机上产生的数据文件要到另外的计算机系统上去读，则应该采用格式输出，并用格式读入. 只有在同一计算机系统中才适合采用无格式读写. 这是因为各计算机系统对数据采用的二进制代码形式有时不同.

§12.2 OPEN 语句和 CLOSE 语句

在向文件输出数据，或者从文件读入数据之前，必须把 WRITE 或 READ 语句中的设备号与具体的文件连接起来. 在 FORTRAN 程序中，这可通过 **OPEN** 语句来实现. 因此，通常也把这一操作称为打开一个文件；除此之外，OPEN 语句还具有其他一些功能. OPEN 语句的形式如下：

```
OPEN(olist)
```

此处 olist 包含以下九个说明符：

UNIT = u　或者直接写 u

FILE = fn

STATUS = st

ACCESS = acc

FORM = fm

RECL = len

BLANK = bl

ERR = s

IOSTAT = ios

以上各说明符除了第一个是必不可少的以外，其他各项都可根据实际需要任意选用.

1. UNIT = u. 此处 u 是一个设备号，它与程序中 WRITE 或 READ 语句中指定的设备号相同. 当此说明符是 OPEN 语句中的第一个说明符时，UNIT = 可以省略. 在一个 OPEN 语句中，只能包含一个设备号，也就是说，一条 OPEN 语句只对应于一个文件.

2. FILE = fn. fn 是字符串表达式，代表文件名（不包括字符串尾部空格）. 当此说明符省略时，由计算机系统来给出文件名. 文件名的形式根据各计算机系统的规定来决定，例如在一般的微型机上可写：

```
OPEN(3,FILE='A:DATA1. DAT')
```

此语句规定把设备号 3 与磁盘驱动器 A 上的名为 DATA1. DAT 的文件连接起来. 在此之后若有语句：

```
WRITE(3,'(4F10.3)')A,B,C,D
```

将把 A，B，C，D 的值按格式写到 A 驱动器磁盘上名为 DATA1. DAT 的文件上去. 在此 OPEN 语句之后，任何用设备号 3 的输入输出都将从该文件中去读或写.

3. STATUS = st. st 是由用户给出的字符串表达式. 此字符串表达式不计尾随空格，应该给出的值为以下四种之一：OLD、NEW、SCRATCH 或 UNKNOWN.

当指明 NEW 时，则表示所指定的文件名是一个以前不存在的新的文件名，如以下语句：

```
OPEN(3,FILE='A:DATA1. DAT',STATUS='NEW')
```

使计算机自动在 A 驱动器的磁盘上建立一个名为 DATA1. DAT 的文件，这个文件是由此 OPEN 语句所新建立的，此时文件的内容是空的，而且文件的状态已从 NEW 变成了 OLD. 如果在此磁盘上已有一个同名的文件存在，则在执行这条语句时将出错，这时应该指定 OLD.

如果 st 为 SCRATCH，那么将由计算机系统为指定的设备号连接一个特殊的‘无名’文件（实际上是由计算机系统给出名字，而不是用户给出名字）. 当程序结束时，或执行与此设备号连接的 CLOSE 语句时，此文件将自动消除. 因此，这种文件只是在程序执行期间临时存在. 显然，在 OPEN 语句中，不能同时指定文件名和 SCRATCH 说明符，因此以下语句是错误的.

```
OPEN(3,FILE='A:DATA1. DAT',STATUS='SCRATCH')
```

只能写成：

```
OPEN(3,STATUS='SCRATCH')
```

若 st 为 UNKNOWN，则各计算机系统将根据本系统的规定来确定指的是以上状态中的哪一种. 当省略 STATUS = st 这一说明符时，则隐含为指定的是 UNKNOWN.

4. ACCESS = acc. acc 是由用户指定的字符表达式. 此字符表达式的值去掉尾部空格必须是

SEQUENTIAL(顺序的)或 DIRECT(直接的). 它指定连接文件的存取方式是顺序的或是直接的. 若此说明符省略,则隐含值为 SEQUENTIAL. 对于一个已存在的文件进行直接存取时,此文件必须是允许直接存取的,否则将出错.

5. FORM = fm. fm 是字符表达式,去掉尾部空格后,应由用户给出此表达式的值是 FORMATTED 或者是 UNFORMATTED. 它意味着指明文件是有格式的或是无格式的. 如果不给出此说明符,则对顺序文件隐含的值是 FORMATTED(有格式的),对直接存取文件隐含的值是 UNFORMATTED(无格式的). 例如:

```
OPEN(2)
```

用来打开设备号 2 上的一个格式顺序文件,此文件的名字由计算机系统自动确定. 语句

```
OPEN(25,FORM = 'UNFORMATTED')
```

用来打开设备号 25 上的一个无格式顺序存取的无名文件. 语句

```
OPEN(9,ACCESS = 'DIRECT',RECL = 130)
```

用来打开设备号 9 上的一个无格式直接存取文件,文件中的记录长度为 130. 语句

```
OPEN(14,FILE = 'MYFILE',STATUS = 'OLD',
$    ACCESS = 'DIRECT',FORM = 'FORMATTED',RECL = 120)
```

用来打开设备号 14 上的一个名为 MYFILE 的有格式的直接存取文件,此文件是一个已存在的文件,其记录长度为 120 字节.

6. RECL = len. len 必须是一个正整数整型表达式,其值指定文件记录的长度. 记录长度单位为字节.

对于一个直接文件,必须给定记录长度. 对于有格式的文件记录,这是允许的最多字符数.

对于顺序文件不必给定记录长度,此说明符必须省略.

7. BLANK = bl. bl 是字符表达式,表达式的值去掉尾部空格后必须是 NULL 或 ZERO. 此值由用户给出,以决定在数值格式输入字段中空格的含义. 若给出 NULL,则意味着除全部是空格的字段为零值外,在指定的设备号上,数值输入字段中的空格全部忽略不计. 若给出 ZERO,则输入字段中的空格处理成零. 当 OPEN 语句中省略此说明符时,则隐含为 NULL.

注意,此说明符不得用于无格式记录文件,而且此说明符只对 OPEN 语句中连接的那个文件起作用. 而对预连接的文件(如打印机、键盘终端、穿卡机等),则由系统设定空格的含义.

8. ERR = s. s 是本程序段中的一条语句标号,当 OPEN 语句执行出错时,转向 s 所指的语句继续执行.

9. IOSTAT = ios. ios 为整型变量或整型数组元素. 当执行含有此说明符的 OPEN 语句时,ios 就有定义. 当执行 OPEN 语句不发生错误时,其值为零;若发生错误,则具有一个正的整数值,具体的数值由计算机系统规定.

使用 OPEN 语句时,了解文件是否已被连接是重要的. 如果已经连接,文件要继续处于同样的状态,只有 BLANK = 说明符可以改动. 如果文件状态要进行其他方面的变动,则应该先用 CLOSE 语句关闭文件,即终止连接,然后再重新打开.

CLOSE 语句. CLOSE 语句称关闭文件语句，用来解除指定设备号与文件的连接. 语句形式是：

CLOSE(olist)

此处 olist 包括以下各项：

UNIT = u （或者直接写 u）

STATUS = st

ERR = s

IOSTAT = ios

在 CLOSE 语句中必须包含一个设备号，也只能包含一个. 其他说明符可根据需要任意选择，其中 ERR = 和 IOSTAT = 两项与 OPEN 语句中的含义相同，因此不再重复.

CLOSE 语句中的 STATUS = st 用来指定文件关闭后是否保留. st 是由用户指定的字符表达式，其值去掉尾部空格后应该为 KEEP（保留）或 DELETE（删除）.

当指定为 KEEP 时，文件被关闭后继续存在，并可被重新连接. 然而当对应的 OPEN 语句中指定文件具有 SCRATCH（清除）性质时，不能使用 KEEP，因为具有 SCRATCH 性质的文件在 CLOSE 以后总是被清除掉的.

当指定为 DELETE 时，文件在关闭后不复存在.

当在 CLOSE 语句中没有给出 STATUS = 这一项说明符时，除了在 OPEN 语句中标明为 SCRATCH 的文件之外，其他文件全部保留.

§12.3 顺序文件和直接文件的存取

顺序文件的存取. 按记录排列顺序依次进行存取的文件称为顺序存取文件. 在用 OPEN 语句打开一个顺序文件后，要么对文件进行写，要么对文件进行读，不允许对顺序文件同时进行读和写. 在 OPEN 语句之后的第一个与之有关的 WRITE 语句总是从文件的第一个记录开始依次输出记录；在 OPEN 语句之后的第一个与之有关的 READ 语句总是从文件的第一个记录开始读入记录.

REWIND 语句（反绕语句）. REWIND 语句只能用于顺序存取文件. 当在 OPEN 语句后，用 WRITE 语句写了一个文件时，想要从头开始再读入此文件各记录的值时，除了可用 CLOSE 语句中断连接，并再用 OPEN 语句重新进行连接之外，还可以直接使用 REWIND 语句，使一个已连接的文件返回到文件的开头，然后从头开始读入记录. 也可以采取同样的方式来重新开始写入记录. REWIND 语句的形式如下：

REWIND(olist)

在此处的 olist 包括以下三项：

1. UNIT = u 或者直接写 u
2. ERR = S
3. IOSTAT = ios

它们的意义与 OPEN 语句中的相似，其中设备号是必须包含的，其他两项可任意选择.

在以下两种情况下存取顺序文件需要使用 REWIND 语句.

1. 重读已写好的文件.

当用 WRITE 语句写完顺序文件之后,可以利用 REWIND 语句,使顺序文件回到开头,然后用 READ 语句重读刚写过的记录.注意,READ 语句的各项细节必须与写此文件的 WRITE 语句一致.

在 READ 语句的变量表中,变量先后的类型必须与对应 WRITE 语句中的一致,个数可以少于 WRITE 语句中变量的个数,这时记录中未读的部分忽略不读.

当利用格式传送时,READ 语句中变量采用的格式和对应的 WRITE 语句中采用的格式应该完全一致.对于用无格式输出的记录,只能用无格式读;输入和输出语句中变量的类型必须一致.

总之必须使读记录的方式与写记录的方式是相容的.例如,可用下面的语句来写顺序存取文件.

```
      DO 66 K=1,6
66    WRITE(1,100)I,J,X,Y
100   FORMAT(2I10,2F10.5)
```

在同一程序单位中可用下面的 READ 语句重新得到这些数据:

```
      REWIND(1)
      DO 77 I=1,6
77    READ(1,100)L,M,A
```

虽然上述 READ 语句少了一个变量,但每次执行 READ 语句时,总是从记录的开头读起,只是每次都不读记录中的最后一个数.

2. 重写顺序文件,然后再读.

当需要重写文件时,先要用 REWIND 语句反绕文件.在重写开始以后(哪怕只重写一个记录),老文件中原有的所有数据全都丢失.

在一个顺序文件上,可以按 WRITE,REWIND,READ 的顺序操作,也可以按 WRITE,REWIND,WRITE 的顺序操作;当然如果是一个老文件,也可以按 READ,REWIND,READ 或 READ,REWIND,WRITE 的顺序进行操作.但决不能出现类似像 WRITE,REWIND,WRITE,READ 这样的操作顺序,也就是说不能在写以后,不经反绕立即开始读,反之亦然.因此,以下的处理过程是错误的:

```
      DO 33 K=1,2
      WRITE(3)K
33    CONTINUE
      READ(3)I
```

注意,如果文件已经回退到开头,再遇到 REWIND 语句时,不发生错误.

BACKSPACE 语句(回退一个记录语句)

BACKSPACE 语句只能用于顺序文件,它使指定设备号连接的文件在当前位置上回退一个记录.语句形式为:

BACKSPACE(olist)

olist 中的项目完全与 REWIND 语句中的相同. 它的使用方式与 REWIND 语句相似. 例如:

```
100   FORMAT(F10.5,F5.2/F10.5)
      WRITE(2,100)X,Y,Z
      BACKSPACE(2)
      READ(2,100)P
```

上述 WRITE 语句将在文件上写入两个记录,READ 语句将读入最后一个记录中变量 *Z* 写入的值.

不能用 BACKSPACE 语句回退一个由表控格式写的记录.

直接文件的存取. 可以对文件中任意一个记录进行直接存取的文件称为直接文件. 程序可以直接读或写文件中的某个记录,而不必从文件的开头依次去读写.

为了达到直接存取的目的,在 FORTRAN77 中规定,相应的 OPEN 语句中必须指定 ACCESS = 'DIRECT'说明符和 REC = 记录长说明符.

在直接存取文件中,由计算机系统规定第一个记录的记录号为 1,第二个记录的记录号为 2,其他依次类推. 当用 READ 或 WRITE 语句读或写直接存取文件时,必须在 READ 或 WRITE 语句后的一对括号中出现说明符:REC = 记录号. 用指定记录号的方法对文件中的记录直接进行存取. 例如:

```
      DIMENSION A(5,5),COL(5)

      WRITE(4,REC=16)A
      READ(4,REC=16)COL
```

上述 WRITE 语句在与设备号 4 连接的直接存取文件中,在第 16 个记录上输出 25 个实型数据,READ 语句则在同一文件的第 16 个记录中读入头 5 个实型数到 COL 数组中,也就是把 A 数组中的第一列数读入 COL 中,其余 20 个数略去不读.

在直接存取文件中,所有记录的长度必须相同.

在程序中,一个打开的文件不可能既用作直接存取,又同时用作顺序存取.

§12.4 程 序 举 例

例 1 分别建立顺序文件和直接存取文件. 从终端输入 N 个学生的姓名和三门课的成绩,分别存入这两个文件中. 文件中每个记录存放一名学生的名字和他的成绩. 假定最多有 100 名学生,每位学生姓名最多由 12 个字符组成.

1. 输入 N 名学生的数据,建立一个名为 SSTUD. DAT 的顺序存取数据文件,存放学生的数据,此文件假定在驱动器 A 的磁盘上. 程序如下:

```
      PROGRAM MAIN
      DIMENSION NAME(100),S(100,3)
```

```
      CHARACTER * 12 NAME
      WRITE( * , * )'INPUT NUMBER OF STUD:'
      READ( * , * )N
      OPEN(4,FILE = 'A:SSTUD. DAT',STATUS = 'NEW')
      DO 10 I = 1,N
      READ( * , * ) NAME(I),(S(I,J),J = 1,3)
10    CONTINUE
      WRITE(4,200)(NAME(I),(S(I,J),J = 1,3),I = 1,N)
200   FORMAT(A12,3F10.2)
      WRITE( *,210)(NAME(I),(S(I,J),J = 1,3),I = 1,N)
210   FORMAT(4X,A12,3F10.2)
      CLOSE(4)
      END
```

2. 输入 N 名学生的数据，建立一个名为 DSTUD. DAT 的直接存取文件，存放学生的数据，此文件假定在驱动器 A 的磁盘上. 程序如下：

```
      PROGRAM MAIN
      DIMENSION NAME(100),S(100,3)
      CHARACTER * 12 NAME
      WRITE( * , * )'INPUT NUMBER OF STUD'
      READ( * , * )N
      DO 10 I = 1,N
      READ( * , * )NAME(I),(S(I,J),J = 1,3)
10    CONTINUE
      WRITE( *,210)(NAME(I),(S(I,J),J = 1,3),I = 1,N)
210   FORMAT(4X,A12,3F10.2)
      OPEN(3,FILE = 'A:DSTUD. DAT',STATUS = 'NEW',
     *     ACCESS = 'DIRECT',FORM = 'FORMATTED',RECL = 50)
      DO 20 I = 1,N
        WRITE(3,200,REC = I)NAME(I),(S(I,J),J = 1,3)
20    CONTINUE
      CLOSE(3)
200   FORMAT(A12,3F10.2)
      END
```

例 **2**　在检查上例输入的学生数据后发现第 K1 个记录中的姓名有错，第 K2 个记录中学生的第二门课程的成绩有错. 编写程序修改 SSTUD. DAT 和 DSTUD. DAT 中的有错数据.

1. 修改顺序存取文件 SSTUD. DAT. 在修改顺序存取文件时，必须从头开始顺序读入每一个记录，把正确的那些记录顺序写到另一个临时文件中；在读入到有错的记录时，则把修改后的内容写到临时文件中. 待全部修改完之后，再把临时文件中的内容顺序传送回 SSTUD. DAT 文件中. 程序如下所示. 程序中与设备号 2 连接的文件是一个临时文件，在程序结束前对应的 CLOSE

语句中规定删除此文件.

```
      PROGRAM MAIN
      DIMENSION A(3)
      CHARACTER*12 NAME,RENAME
      OPEN(1,FILE='A:SSTUD.DAT',STATUS='OLD')
      OPEN(2,FILE='A:TEMP',STATUS='NEW')
      WRITE(*,*)'INPUT NUM OF STUD:'
      READ(*,*)N
      READ(*,*)K1,RENAME
      READ(*,*)K2,S
      DO 10 I=1,N
        READ(1,100)NAME,(A(J),J=1,3)
        IF(I.EQ.K1)THEN
          NAME=RENAME
        ELSE IF(I.EQ.K2)THEN
          A(2)=S
        END IF
        WRITE(2,100)NAME,(A(J),J=1,3)
10    CONTINUE
      REWIND(1)
      REWIND(2)
20    READ(2,100,END=30)NAME,(A(J),J=1,3)
      WRITE(1,100)NAME,(A(J),J=1,3)
      WRITE(*,'(//4X,A,3F10.2)')NAME,(A(J),J=1,3)
      GOTO 20
30    CLOSE(1)
      CLOSE(2,STATUS='DELETE')
100   FORMAT(A12,3F10.2)
      END
```

2. 修改直接存取文件 DSTUD.DAT. 程序如下. 从程序可以看出,修改直接存取文件比修改顺序存取文件简单得多.

```
      PROGRAM MAIN
      DIMENSION A(3)
      CHARACTER*12 NAME,RENAME
      OPEN(2,FILE='A:DSTUD.DAT',STATUS='OLD',ACCESS='DIRECT',
     *     RECL=50,FORM='FORMATTED')
      READ(*,*)K1,RENAME
      READ(*,*)K2,S
      READ(2,100,REC=K1)NAME,(A(I),J=1,3)
      NAME=RENAME
```

```
      WRITE(2,100,REC=K1)NAME,(A(I),I=1,3)
      READ(2,100,REC=K2)NAME,(A(I),I=1,3)
      A(2)=S
      WRITE(2,100,REC=K2)NAME,(A(I),I=1,3)
100   FORMAT(A12,3F10.2)
      CLOSE(2)
      END
```

例 **3**　从学生的数据文件中读入学生的成绩，调用 SORE 子程序来求出学生的平均分，并按成绩优劣排好序. 把排好序的学生数据（姓名、三门课的成绩、平均分），按无格式记录形式放在一个直接存取文件中. 程序清单如下：

```
      PROGRAM T35
      DIMENSION NAME(100),S(100,3),AVE(100)
      CHARACTER*12 NAME,RNAME,FSTAT
      OPEN(1,FILE='A:SSTUD.DAT',STATUS='OLD')
      N=0
2     N=N+1
      READ(1,100,END=4)NAME(N),(S(N,J),J=1,3)
      GOTO 2
4     N=N-1
      CLOSE(1)
      CALL SORE(NAME,S,AVE,N,3)
      OPEN(1,FILE='A:SORT.DAT',STATUS='NEW',
     *     ACCESS='DIRECT',RECL=28)
      DO 10 I=1,N
        WRITE(1) NAME(I),(S(I,J),J=1,3),AVE(I)
10    CONTINUE
      DO 20 I=1,N
        READ(1,REC=I)RNAME,A,B,C,D
        WRITE(*,*)RNAME,A,B,C,D
20    CONTINUE
100   FORMAT(A12,3F10.2)
      CLOSE(1)
      END
```

习　　题

读入 N 位学生的学号和一门课的成绩，分别存入顺序文件和随机存取文件. 然后对这两个文件分别进行以下各项操作.

1. 使文件内容按学生成绩优劣排序.
2. 插入 K1 个记录，使插入后的文件内容仍按序排列.
3. 删除 K2 个记录，使删除后的文件内容仍按序排列.

4. 修改 K3 个记录内容.

5. 把文件中凡是超过平均分的学生成绩打印出来(这时,文件中的记录个数未知).

注意,建立的学生数据文件的记录均为格式记录. 在进行以上操作时,所需临时文件的记录均为无格式记录.

附　录

附录 I　FORTRAN77 与 FORTRAN66 的比较

制定 FORTRAN77 标准时，考虑的一条重要原则是与 1966 标准兼容. 但并不是说，按 FORTRAN66 标准编写的源程序，都可以原封不动地拿来用 FORTRAN77 编译系统去编译它. 它们之间有一些矛盾，应加以注意. 尤其在第一次用 FORTRAN77 编译系统编译 FORTRAN66 源程序时，应细心检查结果是否正确.

FORTRAN77 与 FORTRAN66 的矛盾

在 1978 年美国国家标准化协会关于 FORTRAN77 标准文本的附录中，列出了 24 条矛盾，它们大部分对用户影响不大，现只介绍以下几条：

（a）FORTRAN77 中没有 Hollerith 常数和 Hollerith 数据（如 J = 2HAB 或 DATA J/2HAB/的用法是不允许的），它被 CHARACTER 型数据所代替. 在 FORTRAN66 中，Hollerith 常数可以在 DATA 语句和 CALL 语句中出现，而且字符数据能用 A 格式输入到数值变量中去.

（b）FORTRAN77 对数组元素的每一维下标都要进行检查，不允许超过数组说明中相应的上界. 如：

```
REAL X(10,10)

Y = X(15,2)
```

是不允许的. 但 1966 标准允许，只要不超过此数组的最大下标值即可.

（c）FORTRAN77 不允许从 DO 循环体的外部转到循环体的内部. 而 FORTRAN66 允许在某一特定条件下控制转移到循环体内，即允许循环扩充域.

（d）FORTRAN66 中提供的函数有两类，称为“内部函数”和“基本外部函数”. 内部函数名不允许作为实在变量使用，如果基本外部函数名作为实在变量使用，则名字必须出现在 EXTERNAL 语句中. 在 FORTRAN77 中，所有提供的函数均称为内部函数，并且，除类型转换函数、串比较、选最大值和最小值的函数名不可以作实在变量使用外，其他均可作实在变量使用. 如果内部函数名作为实在变量使用，则它必须出现在 INTRINSIC 语句中，而不是 EXTERNAL 语句中.

（e）增加了更多的内部函数，并分一般名（属名）和专用名. 这些名字的使用和 FORTRAN66 中名字的使用有不一致的地方.

（f）FORTRAN77 指定了内部函数的自变量及结果的单位和范围，而 FORTRAN66 就没有. 因此，FORTRAN66 执行过程中所使用的说明，很可能和 FORTRAN77 标准有些差别.

（g）FORTRAN77 规定顺序存取文件不能同时包含有格式记录和无格式记录，而在 FORTRAN66 中没有规定.

FORTRAN77 对 FORTRAN66 的扩充

FORTRAN77 比 FORTRAN66 增加了许多重要特性，目的是减少对用户的限制. 增加的重要内容有：

（a）CHARACTER 类型说明语句，增加了字符变量和字符处理的功能；

（b）PARAMETER 语句；

（c）块 IF，ELSE，ELSE IF 和 END IF 语句；

（d）直接存取文件；

（e）OPEN 和 CLOSE 语句；

（f）在输入/输出语句中增加了错误条件和文件结束说明符；

（g）增加了撇号（'）编辑，位置编辑（T，TL，TR），符号编辑（S，SP，SS）和空格编辑（BN，BZ）；

（h）增加了直接表输入/输出.

对 **FORTRAN66** 特性的重要改进

（a）可以指定数组的下限，下限可以是负数和 0，66 标准不给出下限，而规定下限为 1；

（b）下标表达式可以是任何整型表达式，FORTRAN66 只能是线性整表达式（即 IK + J 形式，其中 I，J 是整常数，K 是整变量）；

（c）允许在表达式中对不同类型的量进行混合运算，66 标准不允许不同类型的量进行混合运算；

（d）DO 语句中的循环参量（初值、终值、增量）允许是算术表达式，66 文本规定必须是整常数或整变量；

（e）DATA 语句允许使用隐式循环，66 文本不允许；

（f）在输出表中可以有表达式，66 文本不允许；

（g）定义了文件结束记录的读入，66 文本没有明确定义；

（h）执行 DO 循环的重要改变是当循环次数为 0 时，则不执行，而大多数 66 文本不管循环参量为何值时都至少执行一次循环体；

（i）字符类型

CHARACTER 型的引入不仅可以代替 H 型常数和数据，而且也定义了字符数据和数值数据的存储单元. 即：

（1）一个整型、实型或逻辑型数据占一个存储单元；

（2）一个双精度型和复型数据占二个存储单元；

（3）字符数据的每个字符占一个字符存储单元（一般一个字符占一个字节）.

77 标准没有规定数值存储单元和字符存储单元之间的关系. 因此

（1）数值和字符数据不能放在同一公用块中；

（2）数值量不能和字符量等价.

附录Ⅱ　FORTRAN77 内部函数

功　能	属名	专用名	参数个数	类型	
				参数	函数
转换成整型	INT	— INT IFIX IDINT —	1	i r r d c	i i i i i
转换成实型	REAL	REAL FLOAT — SNGL —	1	i i r d c	r r r r r
转换成双精度	DBLE	— — — —	1	i r d c	d d d d
转换成复型	CMPLX	— — — —	1	c r d c	c c c c
转换成字符		CHAR	1	i	c
转换成整数		ICHAR	1	c	i
截去小数部分	AINT	AINT DINT	1	r d	r d
舍入到最接近的整数	ANINT	ANINT DNINT	1	r d	r d
舍入到最接近的整型数	NINT	NINT IDNINT	1	r d	i i
取绝对值	ABS	IABS ABS DABS CABS	1	i r d c	i r d r
求余	MOD	MOD AMOD DMOD	2	i r d	i r d

功　能	属名	专用名	参数个数	类型	
				参数	函数
符号传送	SIGN	ISIGN SIGN DSIGN	2	i r d	i r d
正差	DIM	IDIM DIM DDIM	2	i r d	i r d
双精度乘		DPROD	2	r	d
求虚部		AIMAG	1	c	r
求共轭		CONJG	1	c	c
选最大值	MAX	MAX0 AMAX1 DMAX1	≥2	i r d	i r d
		AMAX0 MAX1		i r	r i
选最小值	MIN	MIN0 AMIN1 DMIN1	≥2	i r d	i r d
		AMIN0 MIN1		i r	r i
平方根	SQRT	SQRT DSQRT CSQRT	1	r d c	r d c
指数	EXP	EXP DEXP CEXP	1	r d c	r d c
自然对数	LOG	ALOG DLOG CLOG	1	r d c	r d c
常用对数	LOG10	ALOG10 DLOG10	1	r d	r d

续表

功　能	属名	专用名	参数个数	类型	
				参数	函数
正弦	SIN	SIN DSIN CSIN	1	r d c	r d c
余弦	COS	COS DCOS CCOS	1	r d c	r d c
正切	TAN	TAN DTAN	1	r d	r d
反正弦	ASIN	ASIN DASIN	1	r d	r d
反余弦	ACOS	ACOS DACOS	1	r d	r d
反正切	ATAN	ATAN DATAN	1	r d	r d
	ATAN2	ATAN2 DATAN2	2	r d	r d

功　能		属名	专用名	参数个数	类型	
					参数	函数
双曲正弦		SINH	SINH DSINH	1	r d	r d
双曲余弦		COSH	COSH DCOSH	1	r d	r d
双曲正切		TANH	TANH DTANH	1	r d	r d
求串长度			LEN	1	ch	i
子串下标			INDEX	2	ch	i
判字符顺序	$ch_1 \geqslant ch_2$		LGE	2	ch	l
	$ch_1 > ch_2$		LGT	2	ch	l
	$ch_1 \leqslant ch_2$		LLE	2	ch	l
	$ch_1 < ch_2$		LLT	2	ch	l

1. 在上述内部函数一览表中，参数和函数值的类型以下述记号表示：

i：整型　　rd：实型或双精度型

r：实型　　ird：整型、实型或双精度型

d：双精度型　　rdc：实型、双精度型或复型

c：复型　　irdc：整型、实型、双精度型或复型

ch：字符型　　L：逻辑型

2. 属名和专用名的说明.

在 FORTRAN77 中，函数专用名的自变量类型有严格要求，必须按规定给出自变量的类型；而在使用函数属名时，只要保证自变量有意义，对类型没有严格要求；因此，函数的属名实际上具有通用的性质.

3. 函数使用中的说明.

（1）CHAR(i). 取 FORTRAN 提供的字符序列中第 i 个字符.

ICHAR(ch)是 CHAR 的逆函数.

（2）MOD(x_1,x_2). 取 x_1 和 x_2 的余数，其中 x_1 是被除数，x_2 是除数.

（3）SIGN(x_1,x_2). 函数值取 x_2 的符号，取 x_1 的绝对值.

（4）DIM(x_1,x_2). 当 $x_1 > x_2$ 时，得 $x_1 - x_2$；当 $x_1 \leqslant x_2$ 时，函数值为 0.

（5）LGE(c_1,c_2). c_1,c_2 均为字符型，当 $c_1 \geqslant c_2$ 时，函数值得‘真’，否则得‘假’. LGT，LLE，LLT 函数的用法相似.

（6）所有三角函数的自变量的单位为弧度，反三角函数所得函数值的单位亦为弧度.

（7）SQRT 函数的自变量不得为负数. 求对数函数 LOG，LOG10 的自变量不得小于等于零.

附录Ⅲ　可执行语句和非执行语句表

FORTRAN77 的语句分为可执行语句和非执行语句.

可执行语句：

- 赋值语句
  - 算术赋值语句
  - 逻辑赋值语句
  - 字符赋值语句
  - 语句标号赋值语句(ASSIGN)
- 控制语句
  - 无条件 GO TO 语句
  - 计算 GO TO 语句
  - 赋值 GO TO 语句
  - 块 IF 语句
  - ELSE IF 语句
  - ELSE 语句
  - END IF 语句
  - 逻辑 IF 语句
  - 算术 IF 语句
  - DO 语句
  - CONTINUE 语句
  - STOP 语句
  - PAUSE 语句
  - END 语句
  - CALL 语句
  - RETURN 语句
- 输入/输出语句
  - READ 语句
  - WRITE 语句
  - PRINT 语句
  - OPEN 语句
  - CLOSE 语句
  - INQUIRE 语句
  - BACKSPACE 语句
  - ENDFILE 语句
  - REWIND 语句

非执行语句：

- 说明语句
  - 类型语句
  - DIMENSION 语句
  - COMMON 语句
  - PARAMETER 语句
  - INTRINSIC 语句
  - EXTERNAL 语句
  - SAVE 语句
  - IMPLICIT 语句
  - EQUIVALENCE 语句

- 程序单位语句
  - PROGRAM 语句
  - FUNCTION 语句
  - SUBROUTINE 语句
  - BLOCK DATA 语句
  - ENTRY 语句
- DATA 语句
- FORMAT 语句
- 语句函数语句

附录Ⅳ　程序单位中语句和注释行的顺序

下表取自 FORTRAN77 标准，说明在程序单位中各种语句和注释行所要求的顺序.

<table>
<tr><td rowspan="6">注　释　行</td><td colspan="3">PROGRAM，FUNCTION，SUBROUTINE 或 BLOCK DATA 语句</td></tr>
<tr><td rowspan="2">FORMAT
语句和
ENTRY
语句</td><td rowspan="2">PARAMETER
语　句</td><td>IMPLICIT 语句</td></tr>
<tr><td>其他说明语句</td></tr>
<tr><td rowspan="2">DATA
语　句</td><td>语句函数语句</td></tr>
<tr><td>可执行语句</td></tr>
<tr><td colspan="3">END 语句</td></tr>
</table>

在表中用垂直线分开的各种语句可以交替出现，横线则给定了在程序单位中各种语句“出现的顺序”. 更精确地说：

（a）在程序单位中第一个语句是 PROGRAM 或 FUNCTION，SUBROUTINE，BLOCKDATA 语句.

（b）说明语句（包括 PARAMETER 语句）优先于 DATA 语句和语句函数语句.

（c）IMPLICIT 语句应在除 PARAMETER 以外的其他所有说明语句之前. PARAMETER 语句可以与 IMPLICIT 和其他说明语句交替出现.

（d）语句函数语句在可执行语句之前，在说明语句之后.

（e）DATA 语句可以任意地与语句函数语句和可执行语句交替出现，即，可以出现在说明语句之后 END 语句之前的任何位置.

（f）FORMAT 语句可以出现在程序单位的任何位置.

（g）ENTRY 语句也可以出现在子程序单位的任何位置，但要除去在块 IF 语句和对应的 END IF 语句之间或在 DO 语句和 DO 循环的终止语句之间的范围.

（h）程序单位的最后一个语句必须是 END 语句.

（i）注释行可以出现在程序单位的任何位置，甚至可以出现在第一个语句之前.

附录 V　FORTRAN77 语句形式表

ASSIGN S TO i	标号赋值语句
BACKSPACE u	回退语句
BACKSPACE(alist)	
BLOCK DATA[sub]	数据块子程序语句
CALL sub[([a[,a] …])]	引用子程序语句
CHARACTER[* len[,]]nam[,nam] …	字符型说明语句
CLOSE(cl ist)	文件关闭语句
COMMON[/[cb] /] nlist	公共语句
[[,] /[cb] /nlist] …	
COMPLEX v[,v] …	复型说明语句
CONTINUE	继续语句
DATA nlist/clist/[[,] nlist/clist] …	数据初值语句
DIMENSION a(d) [,a(a(d))] …	数组说明语句
DO s[,] i = e_1 ,e_2 [,e_3]	循环语句
DOUBLE PRECISION v[,v] …	双精度型说明语句
ELSE	否则语句
ELSE IF (e) THEN	ELSE IF 语句
END	结束语句
END IF	END IF 语句
ENDFILE u	文件结束语句
ENDFILE (alist)	
ENTRY cn[([d[,d] …])]	入口语句
EQUIVALENCE(nlist) [,(nlist)] …	等价语句
EXTERNAL proc[,proc] …	外部语句
FORMAT fs	格式语句
fun ([d[,d] …]) = e	语句函数语句
[typ] FUNCTION fun([d[,d] …])	函数子程序语句
GO TO i[[,] (s[,s] …)]	赋值 GOTO 语句
GO TO s	无条件 GOTO 语句
GO TO (s[,s] …) [,] e	计算 GOTO 语句
IF (e) st	逻辑 IF 语句
IF (e) s_1 ,s_2 ,s_3	算术 IF 语句
IF (e) THEN	块 IF 语句
IMPLICIT typ (a[,a] …) [typ,	
(a[,a] …)] …	类型隐含说明语句
INQUIRE(iflist)	按文件询问语句
INQUIRE(iulist)	按设备询问语句
INTEGER v[,v] …	整型说明语句

INTRINSIC fun[,fun]…	内部语句
LOGICAL v[,v]…	逻辑型说明语句
OPEN(olist)	文件打开语句
PARAMETER(p=e[,p=e]…)	参数语句
PAUSE[n]	暂停语句
PRINT f[,iolist]	打印语句
PROGRAM pgm	程序语句
READ (cilist)[iolist]	读语句
READf[,iolist]	
REAL v[,v]…	实型说明语句
RETURN[e]	返回语句
REWIND u	反绕语句
REWIND(alist)	
SAVE[a[,a]…]	保存语句
STOP[n]	停止语句
SUBROUTINE sub[([d[,d[…]])]	子例行子程序语句
V=e	数值赋值语句
V=e	逻辑赋值语句
V=e	字符赋值语句
WRITE(cilist)[iolist]	输出语句

附录Ⅵ　字符—ASCII 代码—EBCDIC 代码对照表

字　符	ASCII 码		EBCDIC 码	
	八进制数	十六进制数	八进制数	十六进制数
空格	040	20	100	40
!	041	21	132	5A
	042	22	177	7F
#	043	23	173	7B
$	044	24	133	5B
%	045	25	154	6C
&	046	26	120	50
′	047	27	175	7D
(	050	28	115	4D
)	051	29	135	5D
*	052	2A	134	5C
+	053	2B	116	4E
’	054	2C	153	6B
–	055	2D	155	6D
•	056	2E	113	4B
/	057	2F	141	61
0	060	30	360	F0
1	061	31	361	F1
2	062	32	362	F2
3	063	33	363	F3
4	064	34	364	F4
5	065	35	365	F5
6	066	36	366	F6
7	067	37	367	F7
8	070	38	370	F8
9	071	39	371	F9
:	072	3A	172	7A
;	073	3B	136	5E
<	074	3C	114	4C
=	075	3D	176	7E
>	076	3E	156	6E
?	077	3F	157	6F
@	100	40	174	7C
A	101	41	301	C1
B	102	42	302	C2
C	103	43	303	C3
D	104	44	304	C4
E	105	45	305	C5

续表

字　符	ASCII 码		EBCDIC 码	
	八进制数	十六进制数	八进制数	十六进制数
F	106	46	306	C6
G	107	47	307	C7
H	110	48	310	C8
I	111	49	311	C9
J	112	4A	321	D1
K	113	4B	322	D2
L	114	4C	323	D3
M	115	4D	324	D4
N	116	4E	325	D5
O	117	4F	326	D6
P	120	50	327	D7
Q	121	51	330	D8
R	122	52	331	D9
S	123	53	342	E2
T	124	54	343	E3
U	125	55	344	E4
V	126	56	345	E5
W	127	57	346	E6
X	130	58	347	E7
Y	131	59	350	E8
Z	132	5A	351	E9
[	133	5B		
\	134	5C	340	E0
]	135	5D		
^或↑	136	5E		
_（下划线）或←	137	5F		
a	141	61	201	81
b	142	62	202	82
c	143	63	203	83
d	144	64	204	84
e	145	65	205	85
f	146	66	206	86
g	147	67	207	87
h	150	68	210	88
i	151	69	211	89
j	152	6A	221	91
k	153	6B	222	92
l	154	6C	223	93
m	155	6D	224	94

续表

字　符	ASCII 码		EBCDIC 码	
	八进制数	十六进制数	八进制数	十六进制数
n	156	6E	225	95
o	157	6F	226	96
p	160	70	227	97
q	161	71	230	98
r	162	72	231	99
s	163	73	242	A2
t	164	74	243	A3
u	165	75	244	A4
v	166	76	245	A5
w	167	77	246	A6
x	170	78	247	A7
y	171	79	250	A8
z	172	7A	251	A9
{	173	7B	300	C0
|	174	7C	117	4F
}	175	7D	320	D0
~	176	7E		

注：有几个字符无相应的 EBCDIC 代码，本表中该栏空着.

附录Ⅶ　常用基本字符卡片编码表

字　符	卡片穿孔位置	字　符	卡片穿孔位置
0	0	O	11～6
1	1	P	11～7
2	2	Q	11～8
3	3	R	11～9
4	4	S	0～2
5	5	T	0～3
6	6	U	0～4
7	7	V	0～5
8	8	W	0～6
9	9	X	0～7
A	12～1	Y	0～8
B	12～2	Z	0～9
C	12～3	=	6～8
D	12～4	+	12～6～8
E	12～5	-	11
F	12～6	*	11～4～8
G	12～7	/	0～1
H	12～8	(	12～5～8
I	12～9	)	11～5～8
J	11～1	,	0～3～8
K	11～2	.	12～3～8
L	11～3	′	5～8
M	11～4	$	11～3～8
N	11～5	␣（空格）	不穿孔

郑 重 声 明